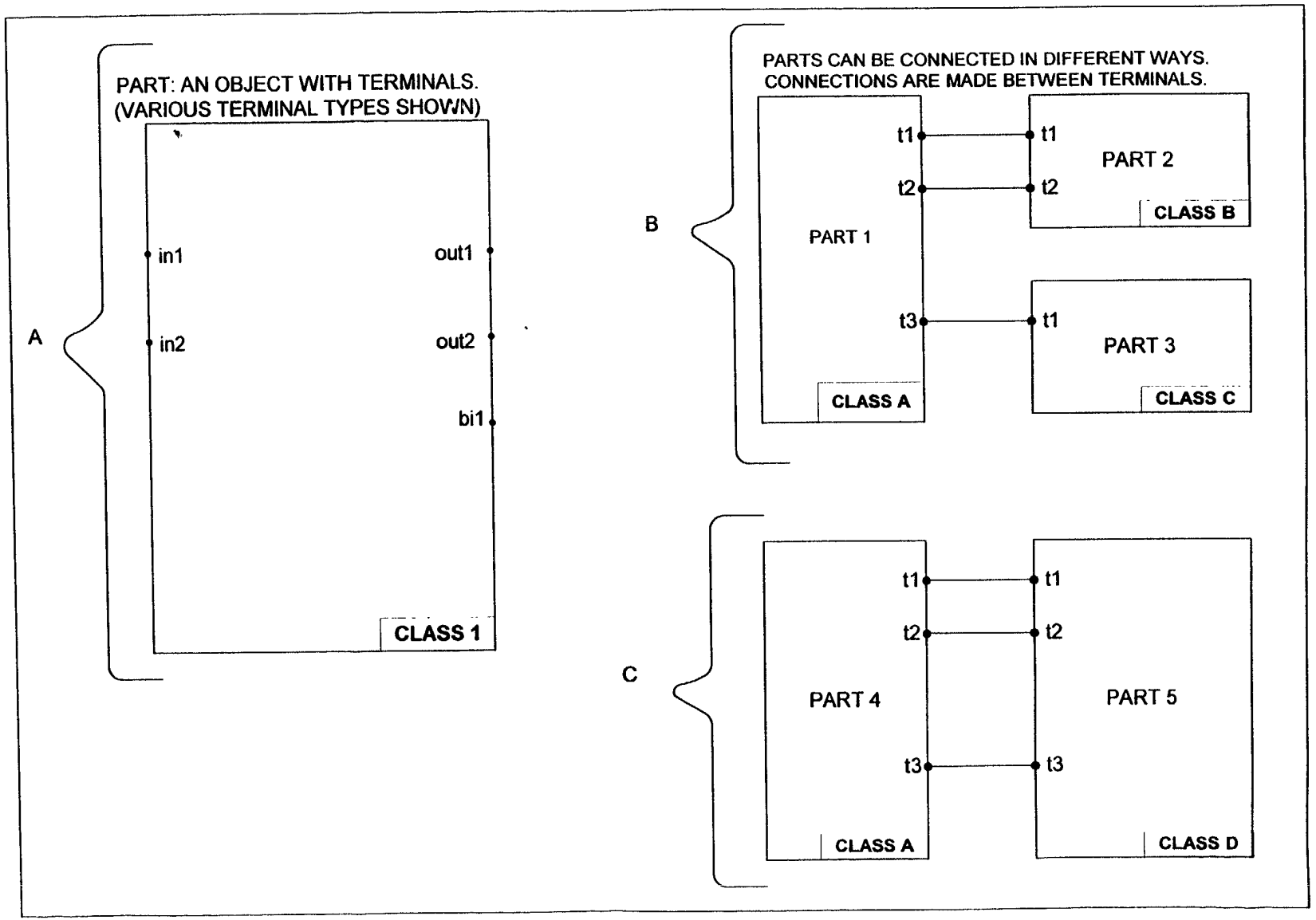

A
PART: AN OBJECT WITH TERMINALS.
(VARIOUS TERMINAL TYPES SHOWN)
in1
in2
out1
out2
bi1
CLASS 1
B
PARTS CAN BE CONNECTED IN DIFFERENT WAYS.
CONNECTIONS ARE MADE BETWEEN TERMINALS.
PART 1
t1
t2
t3
CLASS A
PART 2
t1
t2
CLASS B
PART 3
t1
CLASS C
C
PART 4
t1
t2
t3
CLASS A
PART 5
t1
t2
t3
CLASS D

FIG. 1

Table A: OPERATIONS
ACQUIRE
RELEASE
CONNECT
DISCONNECT
GET INFO
GET CONTEXT
SET CONTEXT
OPEN QUERY
GET FIRST
GET NEXT
GET PREV
GET LAST
CLOSE QUERY

Table B: CONTROL BLOCK (Used with all Operations)	
NAME	STRING
ID	NUMBER
CONNECTION_ID	NUMBER
TYPE	ENUM: FUNCTION, MESSAGE, INTERFACE...
DATA	BINARY WITH SIZE
CONTEXT	NUMBER

Table C: COMMON TERMINAL INFORMATION FIELDS	
NEGOTIATION	ENUM: FIXED, NEGOTIABLE
DIRECTION	ENUM: IN, OUT, BI-DIR, OTHER
SYNCHRONOSITY	ENUM: SYNC, ANBIENT, ASYNC
CARDINALITY	NUMBER
CONTRACT NAME	STRING
CONTRACT ID	BINARY WITH SIZE
(Other Fields can be defined)	

All fields on the terminal info descriptor are returned together in the data field of the control block by GET INFO operation

FIG. 2

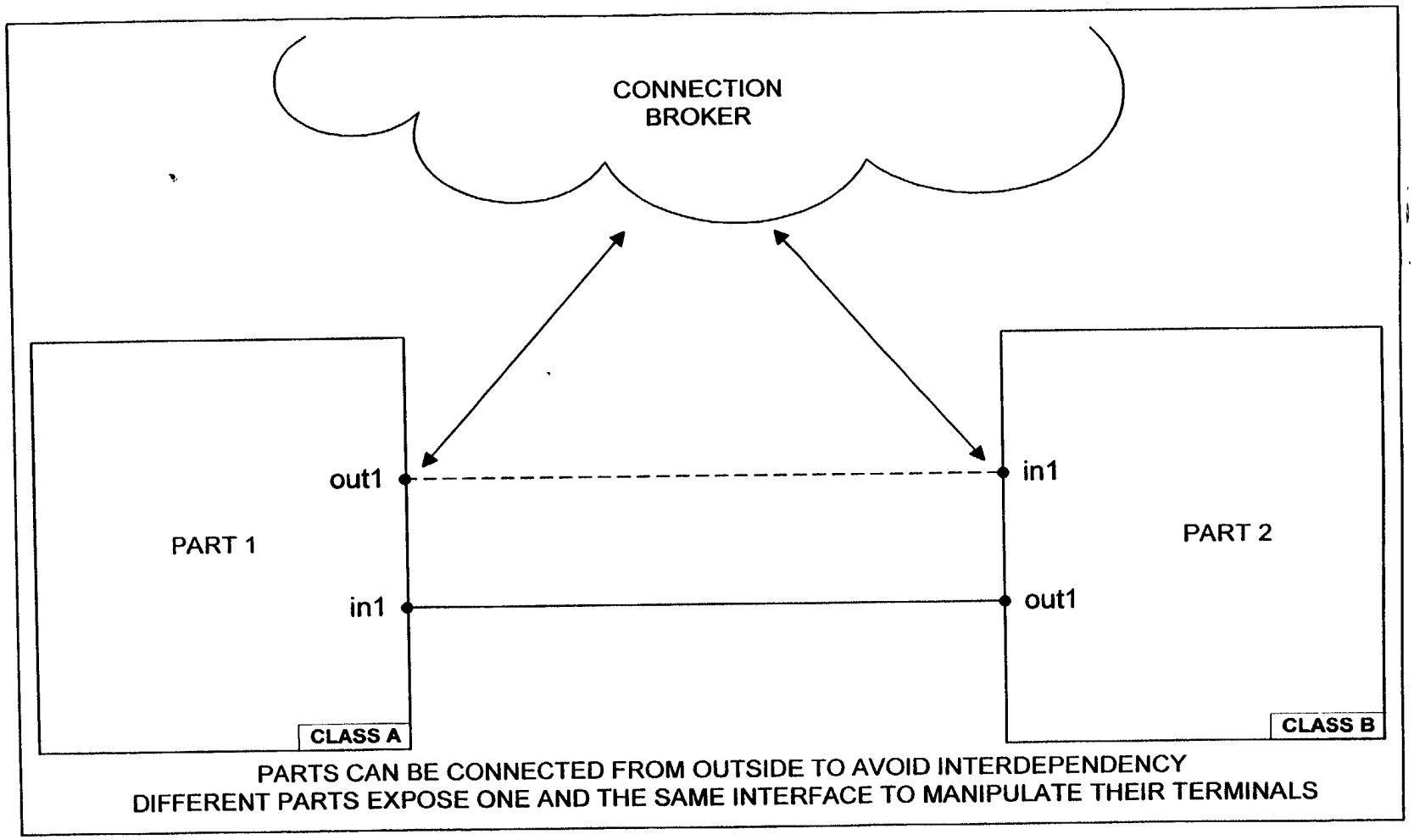

CONNECTION
BROKER
out1
in1
PART 1
PART 2
in1
out1
CLASS A
CLASS B
PARTS CAN BE CONNECTED FROM OUTSIDE TO AVOID INTERDEPENDENCY
DIFFERENT PARTS EXPOSE ONE AND THE SAME INTERFACE TO MANIPULATE THEIR TERMINALS

FIG. 3

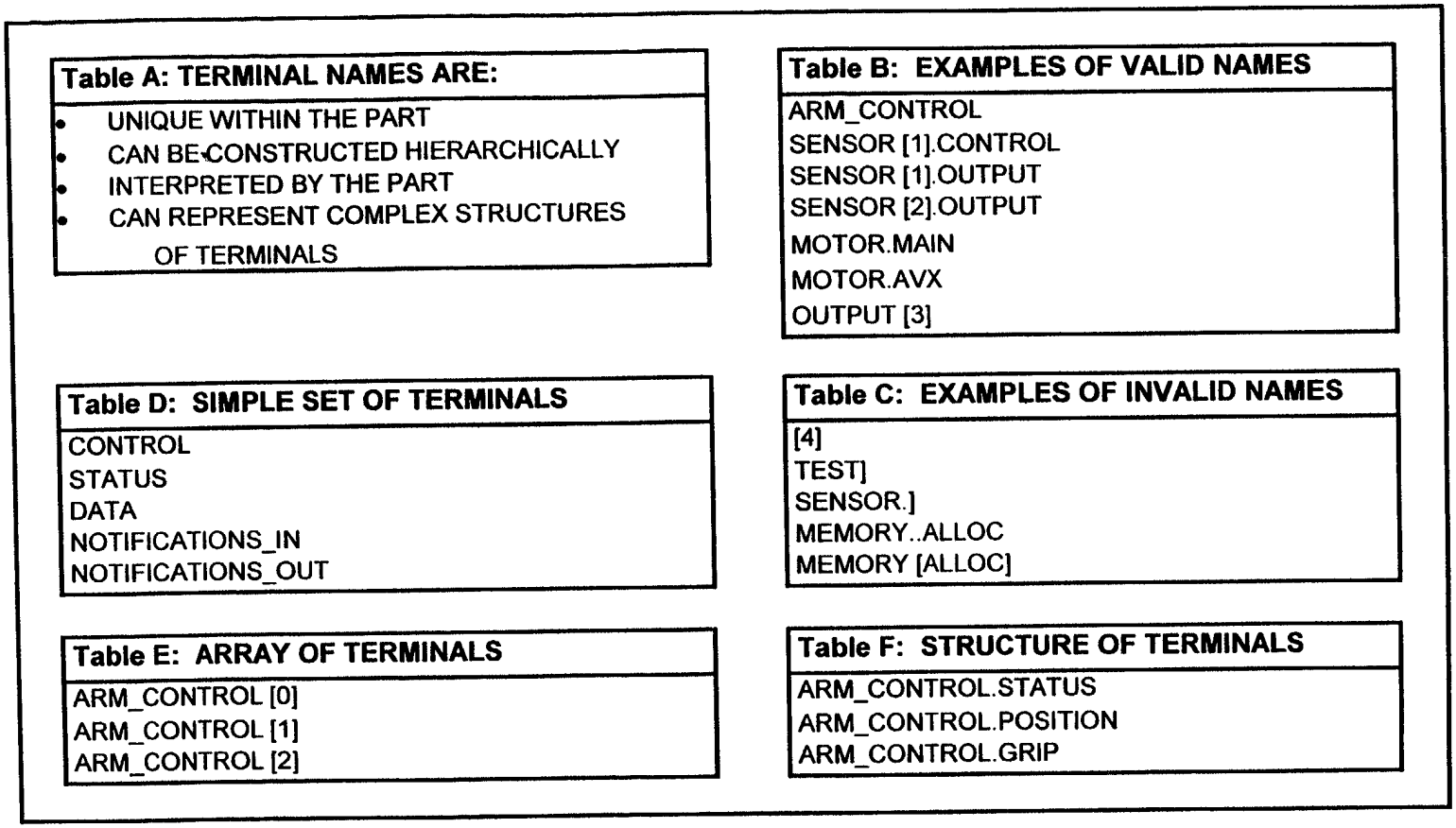


Table A: TERMINAL NAMES ARE:
• UNIQUE WITHIN THE PART
• CAN BE CONSTRUCTED HIERARCHICALLY
• INTERPRETED BY THE PART
• CAN REPRESENT COMPLEX STRUCTURES OF TERMINALS
Table B: EXAMPLES OF VALID NAMES
ARM_CONTROL
SENSOR [1].CONTROL
SENSOR [1].OUTPUT
SENSOR [2].OUTPUT
MOTOR.MAIN
MOTOR.AVX
OUTPUT [3]
Table D: SIMPLE SET OF TERMINALS
CONTROL
STATUS
DATA
NOTIFICATIONS_IN
NOTIFICATIONS_OUT
Table C: EXAMPLES OF INVALID NAMES
[4]
TEST]
SENSOR.]
MEMORY..ALLOC
MEMORY [ALLOC]
Table E: ARRAY OF TERMINALS
ARM_CONTROL [0]
ARM_CONTROL [1]
ARM_CONTROL [2]
Table F: STRUCTURE OF TERMINALS
ARM_CONTROL.STATUS
ARM_CONTROL.POSITION
ARM_CONTROL.GRIP

FIG. 4

TYPE	DATA PROVIDED IN THE DATA FIELD OF CONTROL BLOCK
FUNCTION	POINTER TO FUNCTION, CONTEXT, CONTRACT ID
MESSAGE	OBJECT ID, MESSAGE ID, CONTRACT ID
MESSAGE INTERFACE	OBJECT ID, CONTRACT ID
VTBL INTERFACE	POINTER TO INTERFACE, CONTRACT ID
OLE INTERFACE	POINTER TO INTERFACE, OLE ID
VTBL PLUG	POINTER TO INTERFACE, CONTRACT ID
PIPE	FILE HANDLE, CONTRACT ID

THE DATA ABOVE IS EXCHANGED BETWEEN TERMINALS IN THE PROCESS OF ESTABLISHING OF CONNECTION USING OPERATIONS ACQUIRE AND CONNECT OF THE TERMINAL INTERFACE

FIG. 5

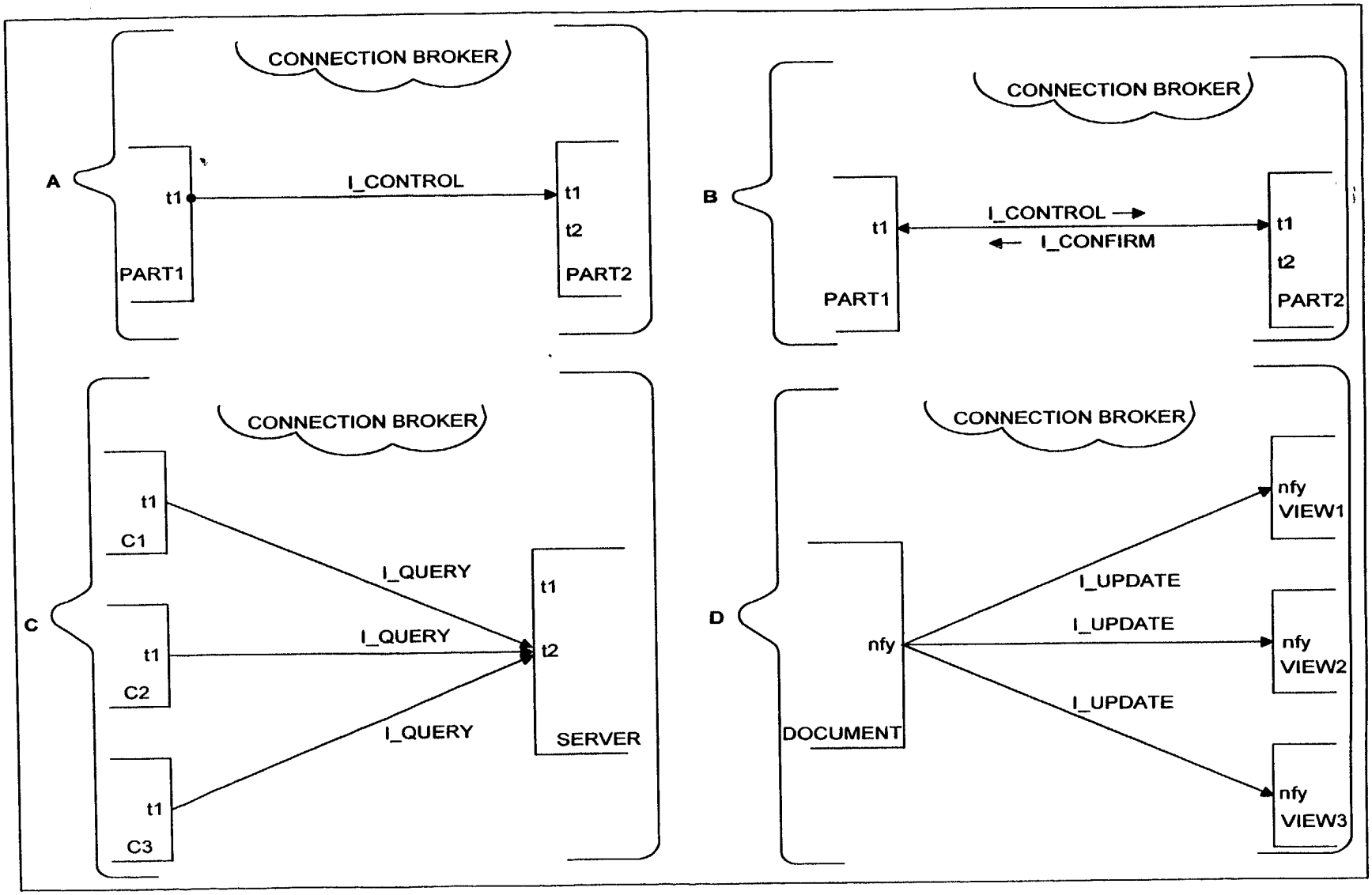

CONNECTION BROKER
A
t1
I_CONTROL
t1
t2
PART1
PART2
CONNECTION BROKER
B
t1
I_CONTROL
I_CONFIRM
t1
t2
PART1
PART2
CONNECTION BROKER
C
t1
C1
t1
C2
t1
C3
I_QUERY
I_QUERY
I_QUERY
t1
t2
SERVER
CONNECTION BROKER
D
nfy
DOCUMENT
I_UPDATE
I_UPDATE
I_UPDATE
nfy
VIEW1
nfy
VIEW2
nfy
VIEW3

FIG. 6

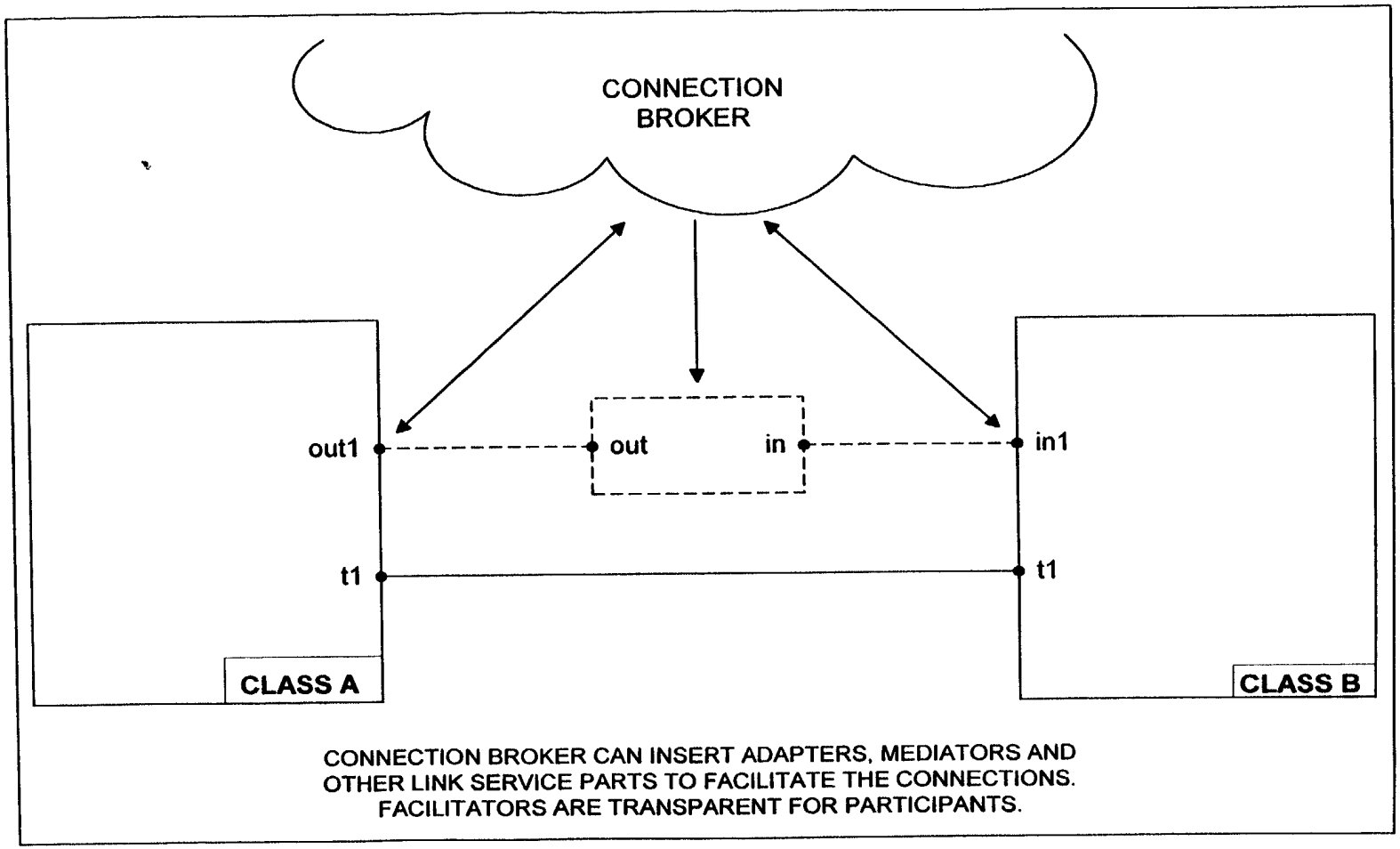


FIG. 7

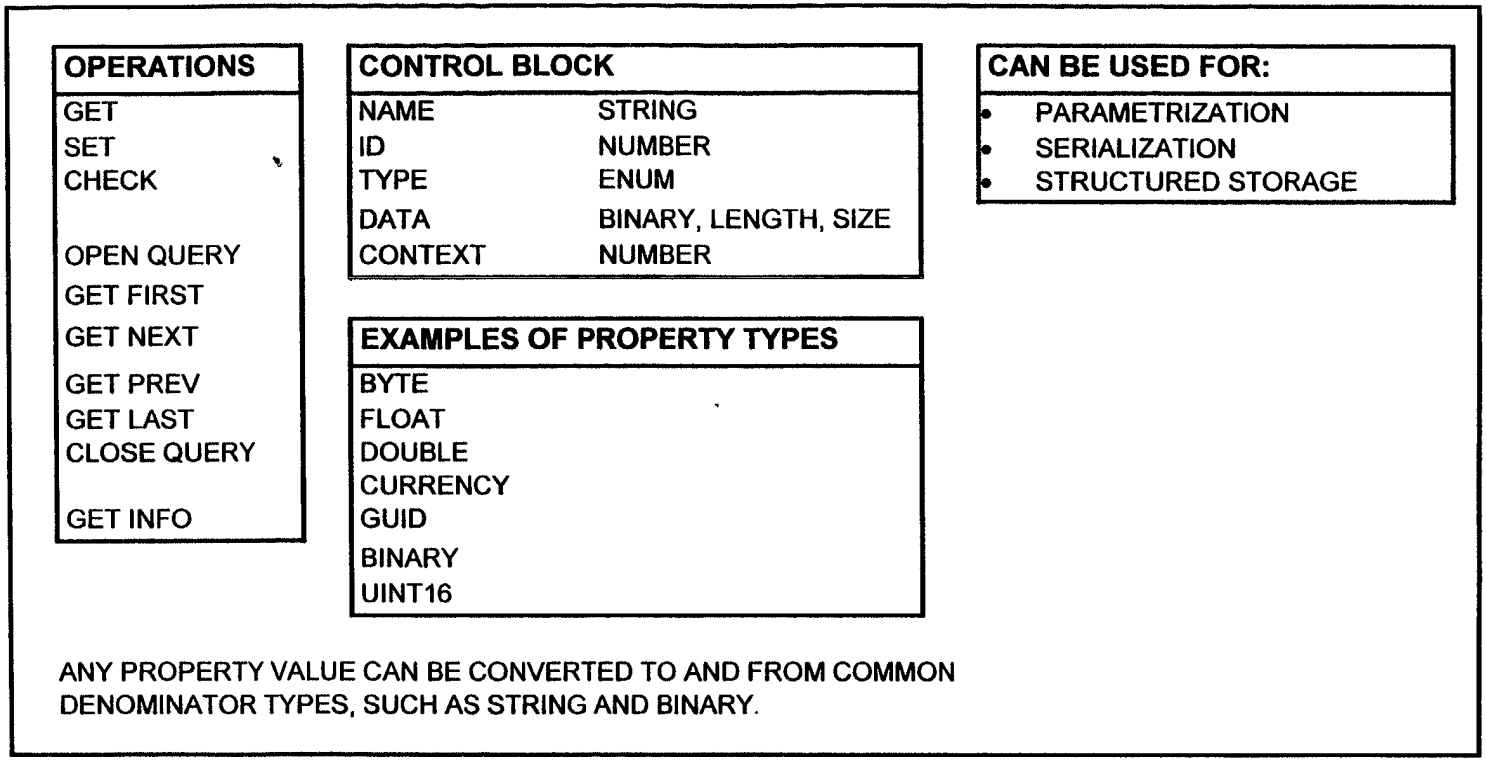

OPERATIONS
GET
SET
CHECK
OPEN QUERY
GET FIRST
GET NEXT
GET PREV
GET LAST
CLOSE QUERY
GET INFO
CONTROL BLOCK
NAME STRING
ID NUMBER
TYPE ENUM
DATA BINARY, LENGTH, SIZE
CONTEXT NUMBER
EXAMPLES OF PROPERTY TYPES
BYTE
FLOAT
DOUBLE
CURRENCY
GUID
BINARY
UINT16
CAN BE USED FOR:
• PARAMETRIZATION
• SERIALIZATION
• STRUCTURED STORAGE
ANY PROPERTY VALUE CAN BE CONVERTED TO AND FROM COMMON DENOMINATOR TYPES, SUCH AS STRING AND BINARY.

FIG. 8

Table A: PROPERTY NAMES ARE:

- UNIQUE WITH THE PART
- CAN BE CONSTRUCTED HIERARCHICALLY
- INTERPRETED BY THE PART
- CAN REPRESENT COMPLEX DATA STRUCTURES
- QUERIES CAN BE OPEN BY PARTIAL NAMES USING WILDCARDS
- OTHER OPERATIONS ON PARTIAL NAMES MEAN OPERATION IS PERFORMED ON A SET OF PROPERTIES THE NAMES OF WHICH SATISFY THE WILDCARD TEMPLATE

Table B: EXAMPLES: VALID AND INVALID NAMES

VALID:	WEIGHT
	DATA [1]
	SENSOR.PRESSURE
INVALID:	READOUT]
	SENSOR.]

Table C: EXAMPLES: PARTIAL NAMES

A.TRAIN [*],CODUCTOR.NAME
B.TRAIN [3].CONDUCTOR.*
C.TRAIN [3].*.NAME

Table D: EXAMPLES: SIMPLE NAMES

BUFFER_SIZE
MAX_BUFFERS
PRIORITY
STORAGE_LOCATION

Table E: EXAMPLES: ARRAYS AND STRUCTURES

READOUT [0]	SENSOR.PRESSURE
READOUT [1]	SENSOR.TEMPERATURE
READOUT [2]	SENSOR.STATE

FIG. 9

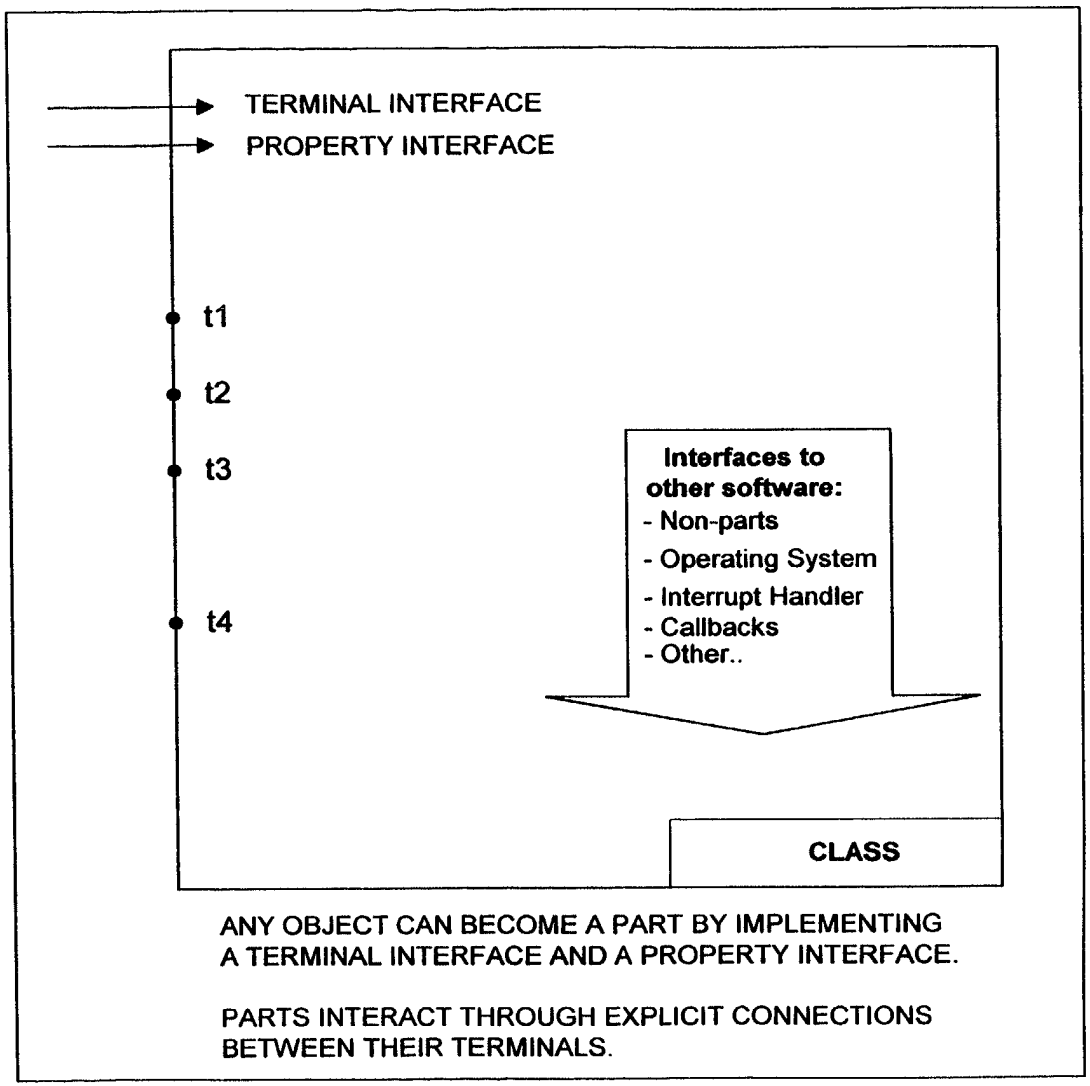

TERMINAL INTERFACE
PROPERTY INTERFACE
t1
t2
t3
t4
Interfaces to other software:
- Non-parts
- Operating System
- Interrupt Handler
- Callbacks
- Other..
CLASS
ANY OBJECT CAN BECOME A PART BY IMPLEMENTING A TERMINAL INTERFACE AND A PROPERTY INTERFACE.
PARTS INTERACT THROUGH EXPLICIT CONNECTIONS BETWEEN THEIR TERMINALS.

FIG. 10

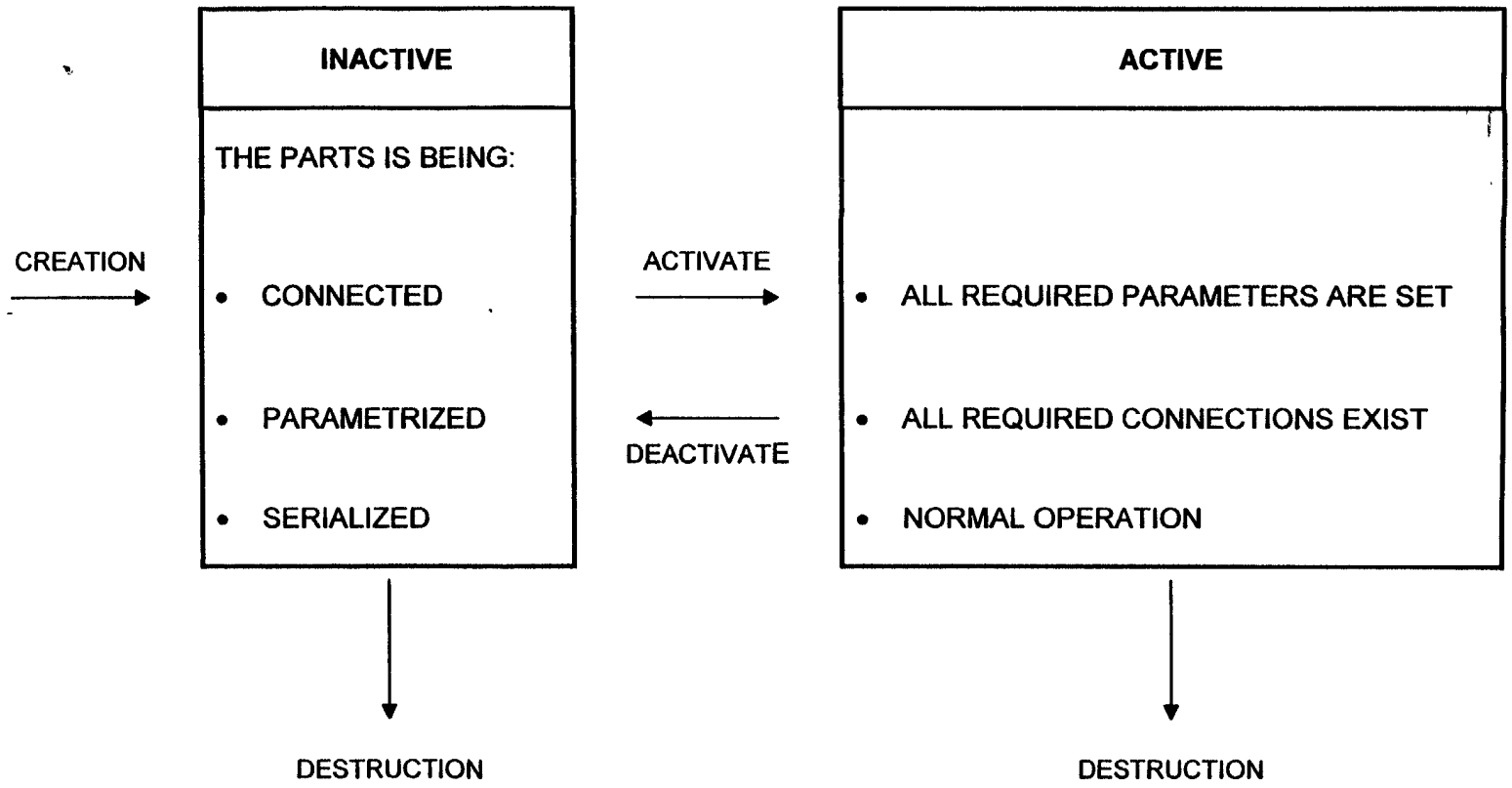
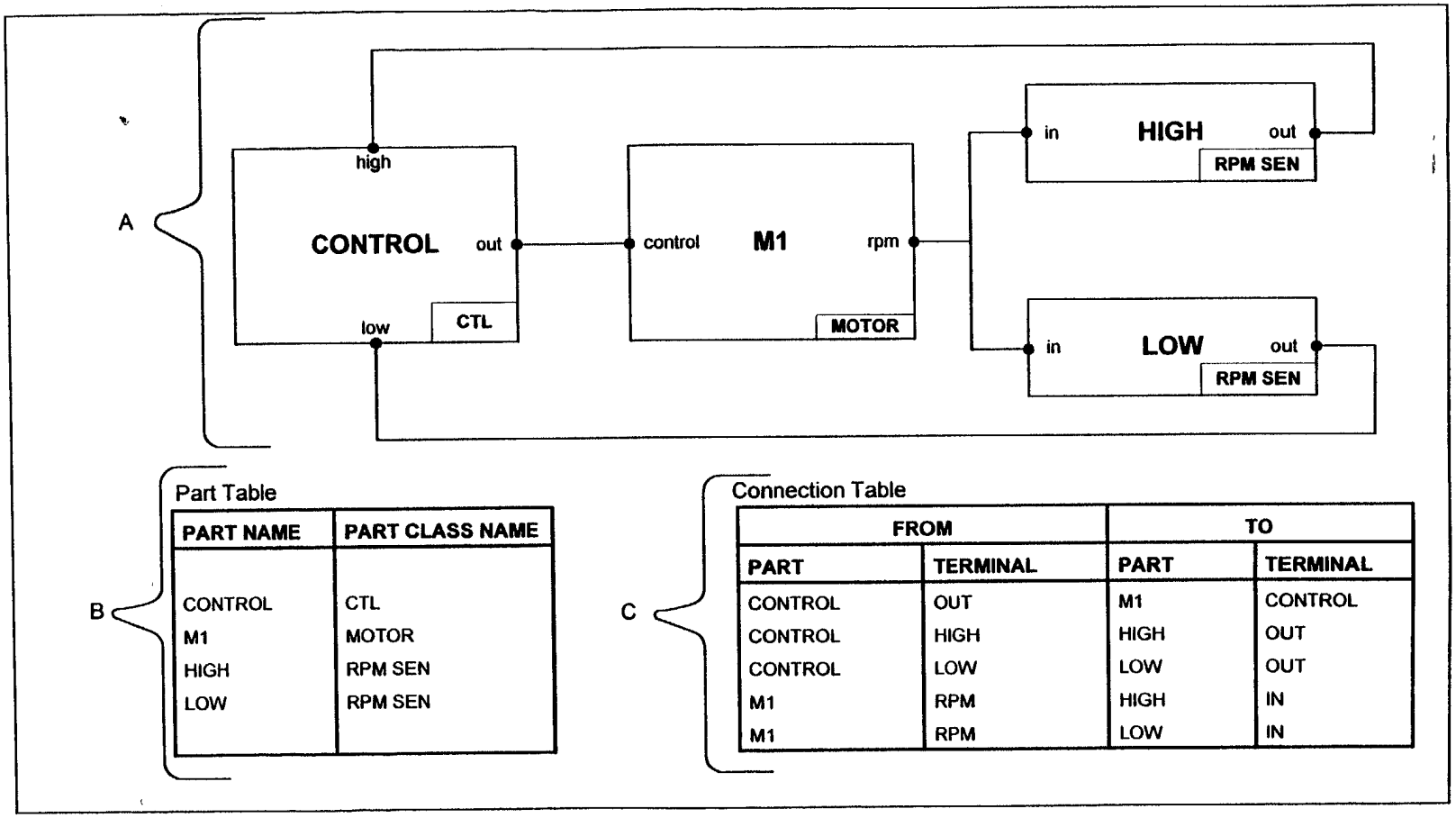

INACTIVE
THE PARTS IS BEING:
CREATION
• CONNECTED
• PARAMETRIZED
• SERIALIZED
DESTRUCTION
ACTIVATE
DEACTIVATE
ACTIVE
• ALL REQUIRED PARAMETERS ARE SET
• ALL REQUIRED CONNECTIONS EXIST
• NORMAL OPERATION
DESTRUCTION

FIG. 11



B

Part Table

PART NAME	PART CLASS NAME
CONTROL	CTL
M1	MOTOR
HIGH	RPM SEN
LOW	RPM SEN

C

Connection Table

FROM		TO	
PART	TERMINAL	PART	TERMINAL
CONTROL	OUT	M1	CONTROL
CONTROL	HIGH	HIGH	OUT
CONTROL	LOW	LOW	OUT
M1	RPM	HIGH	IN
M1	RPM	LOW	IN

FIG. 12

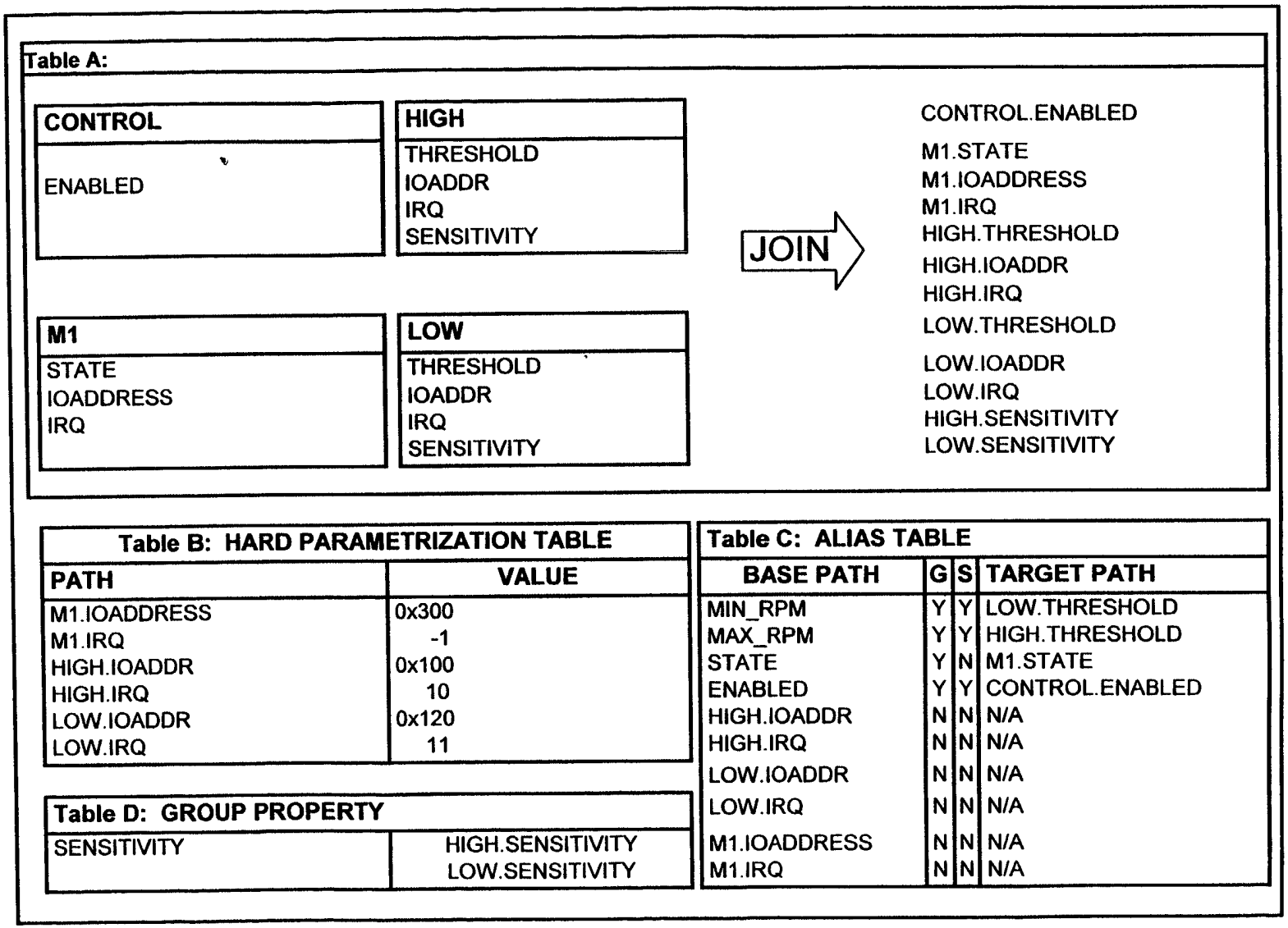


Table B: HARD PARAMETRIZATION TABLE

PATH	VALUE
M1.IOADDRESS	0x300
M1.IRQ	-1
HIGH.IOADDR	0x100
HIGH.IRQ	10
LOW.IOADDR	0x120
LOW.IRQ	11

Table C: ALIAS TABLE

BASE PATH	G	S	TARGET PATH
MIN_RPM	Y	Y	LOW.THRESHOLD
MAX_RPM	Y	Y	HIGH.THRESHOLD
STATE	Y	N	M1.STATE
ENABLED	Y	Y	CONTROL.ENABLED
HIGH.IOADDR	N	N	N/A
HIGH.IRQ	N	N	N/A
LOW.IOADDR	N	N	N/A
LOW.IRQ	N	N	N/A
M1.IOADDRESS	N	N	N/A
M1.IRQ	N	N	N/A

Table D: GROUP PROPERTY

SENSITIVITY	HIGH.SENSITIVITY LOW.SENSITIVITY

FIG. 13

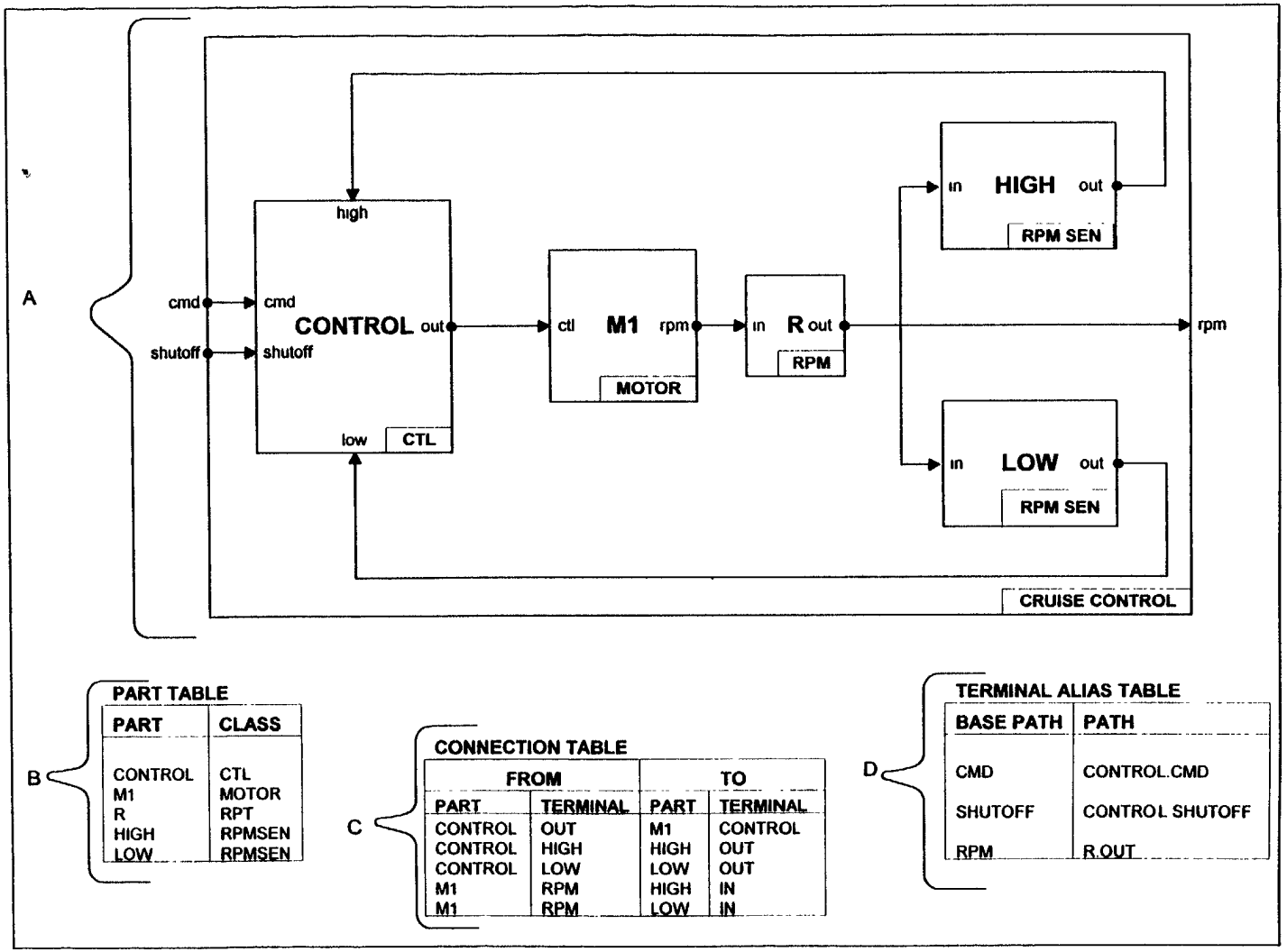

A
high
cmd
cmd
CONTROL
out
shutoff
shutoff
low
CTL
ctl
M1
rpm
MOTOR
in
R
out
RPM
rpm
in
HIGH
out
RPM SEN
in
LOW
out
RPM SEN
CRUISE CONTROL
B
PART TABLE
PART
CLASS
CONTROL
CTL
M1
MOTOR
R
RPT
HIGH
RPMSEN
LOW
RPMSEN
C
CONNECTION TABLE
FROM
TO
PART
TERMINAL
PART
TERMINAL
CONTROL
OUT
M1
CONTROL
CONTROL
HIGH
HIGH
OUT
CONTROL
LOW
LOW
OUT
M1
RPM
HIGH
IN
M1
RPM
LOW
IN
D
TERMINAL ALIAS TABLE
BASE PATH
PATH
CMD
CONTROL.CMD
SHUTOFF
CONTROL.SHUTOFF
RPM
R.OUT

FIG. 14

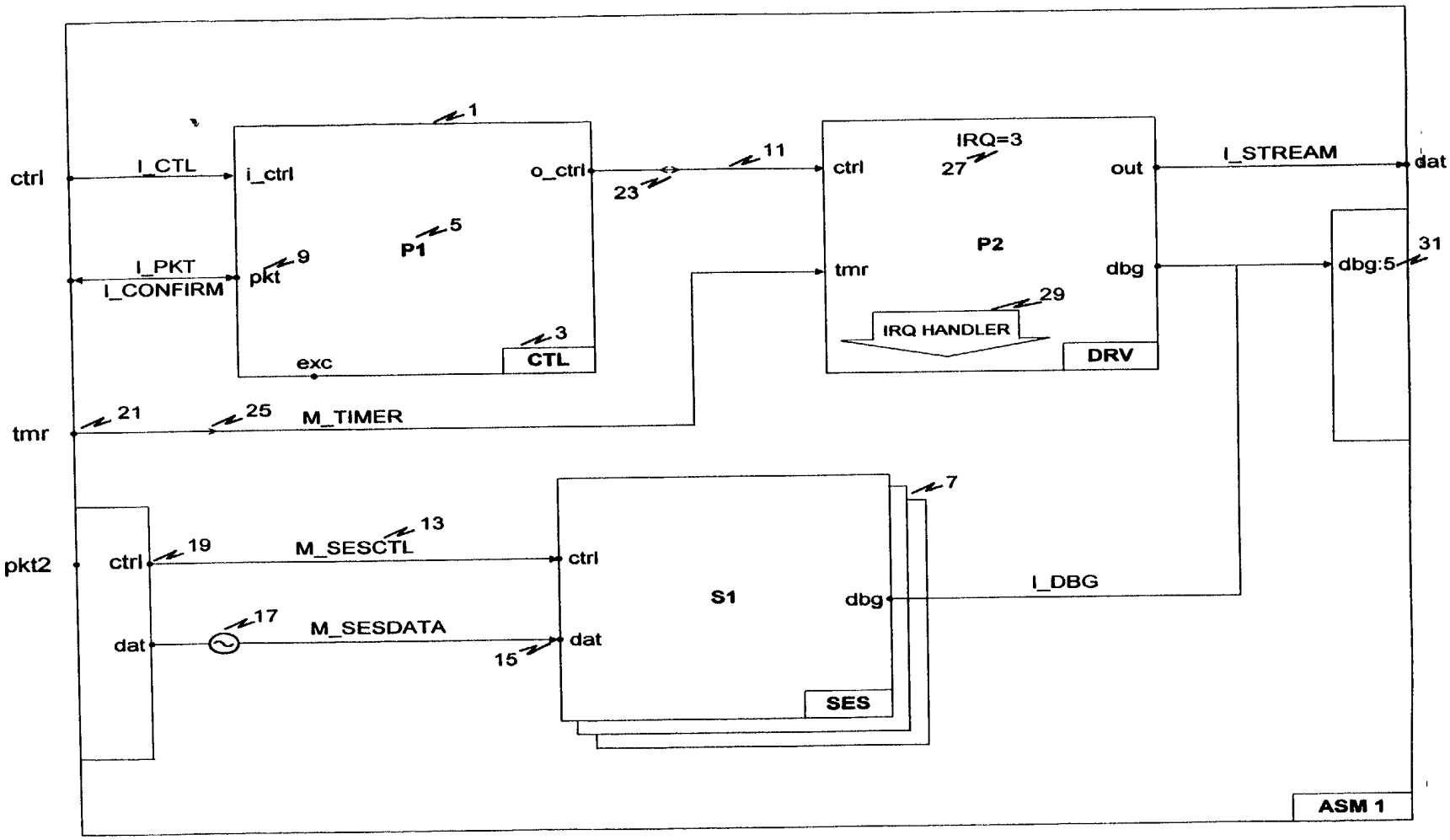

ctrl
I_CTL
i_ctrl
o_ctrl
1
11
23
P1
5
I_PKT
I_CONFIRM
pkt
9
exc
3
CTL
IRQ=3
27
ctrl
out
I_STREAM
dat
P2
tmr
dbg
dbg:5
31
29
IRQ HANDLER
DRV
tmr
21
25
M_TIMER
pkt2
ctrl
19
M_SESCTL
13
ctrl
S1
7
dbg
I_DBG
dat
17
M_SESDATA
dat
15
SES
ASM 1

FIG. 15

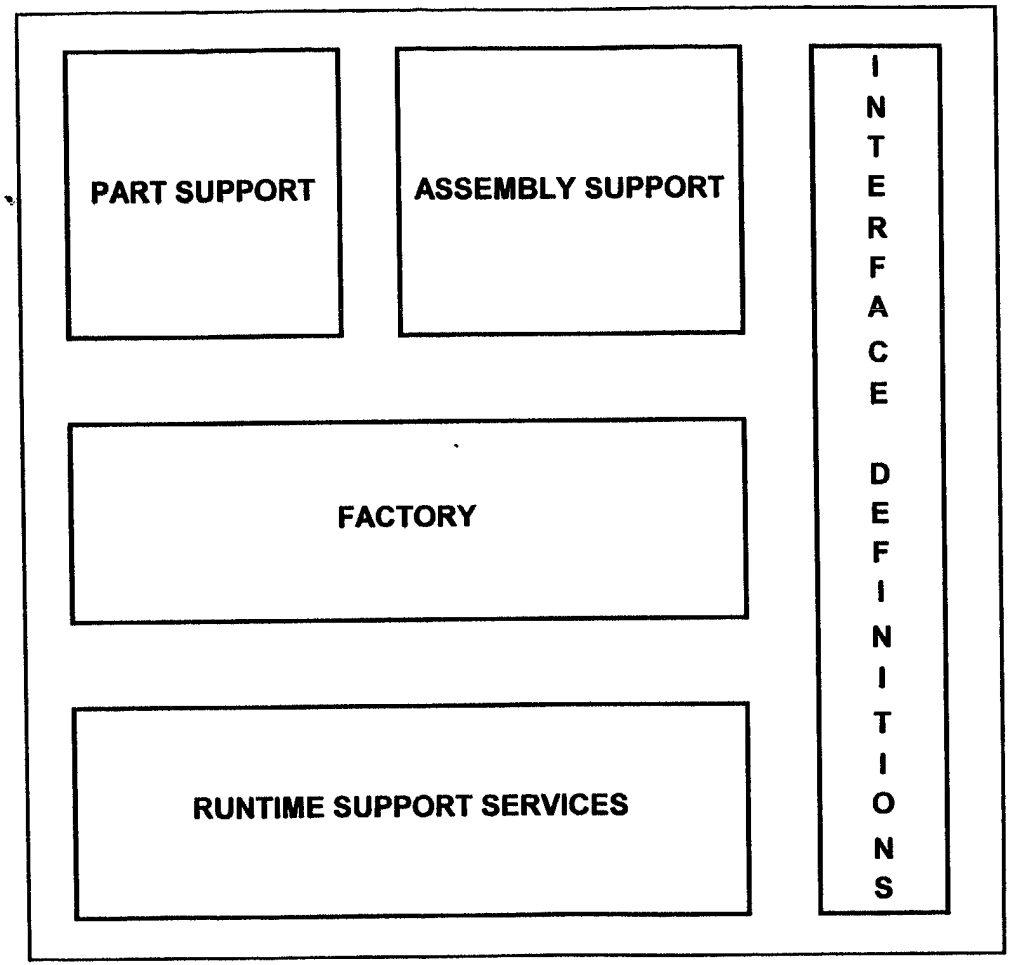

PART SUPPORT
ASSEMBLY SUPPORT
INTERFACE DEFINITIONS
FACTORY
RUNTIME SUPPORT SERVICES

FIG. 16

```
1   /* ------------------------------------------------------------------------ */
2   /*                          ETC: Global system definitions                  */
3   /*                                                                          */
4   /*                      TYPES.H - Portable type definitions                 */
5   /* ------------------------------------------------------------------------ */
6
7   #ifndef _TYPES_DEFINED
8   #define _TYPES_DEFINED
9
10  #if (!defined(BIT32) && !defined(BIT16))
11     #error BIT16 or BIT32 must be defined
12  #endif
13
14  /* --- Common types ---------------------------------------------------- */
15
16  typedef int  bool;
17
18  #ifndef TRUE
19    #define FALSE     ((bool) 0)
20    #define TRUE      ((bool) (!FALSE))
21  #endif
22
23  typedef signed   char   schar;
24  typedef unsigned char   uchar;
25
26  #ifndef _WINNT_
27     #define MINCHAR  ((char) 0x00)          // 0x80 in WINNT.H!!
28     #define MAXCHAR  ((char) 0x7F)
29  #endif
30
31  #define MINSCHAR ((char) 0x80)
32  #define MAXSCHAR ((char) 0x7F)
33
34  #define MINUCHAR ((char) 0x00)
35  #define MAXUCHAR ((char) 0xFF)
36
37  typedef void   *_datap;
38  typedef void   *_codep;  //$$ "void *_codep (void);" would not work
39
40  #ifdef __cplusplus
41     #define NULLD    0
42     #define NULLC    0
43  #else
44     #define NULLD ((_datap) 0)
45     #define NULLC ((_codep) 0)
46  #endif
47
48  #ifndef NULL
49  #define NULL ((void *) 0)
50  #endif
51
52  /*
53    Notes:
54
55    1.  Types _datap and _codep are provided for systems/models in which
56        data pointers are of different size than code pointers.  See also
57        type _pointer in the compatibility types section of this header file.
58  */
59
```

FIG. 17A

```
/* --- Machine types ------------------------------------------------------ */

typedef unsigned       char byte;
typedef unsigned short int  word;
typedef unsigned long  int  dword;

#define MINBYTE  ((byte)  0x00)
#ifndef _WINNT_
   #define MAXBYTE  ((byte)  0xFF)
#endif

#define MINWORD  ((word)  0x0000U)
#ifndef _WINNT_
   #define MAXWORD  ((word)  0xFFFFU)
#endif

#define MINDWORD ((dword) 0x00000000UL)
#ifndef _WINNT_
   #define MAXDWORD ((dword) 0xFFFFFFFFUL)
#endif

/* --- Tolerance types (hardware-dependent) ------------------------------- */

typedef unsigned int  uint;     //guaranteed to be at least 16 bits wide.
typedef signed   int  sint;     //guaranteed to be at least 16 bits wide.

#define MININT  (( int) 0x0000)
#define MAXINT  (( int) 0x7FFF)

#define MINSINT ((sint) -32767-1)
#define MAXSINT ((sint) 0x7FFF)

#define MINUINT ((uint) 0x0000U)
#define MAXUINT ((uint) 0xFFFFU)

/*
  Notes:

   1.  Use tolerance types for variables that are expected to only take
       small values which will never exceed the minimum number of guaranteed
       bit width, so that the compiler can choose the actual type which the
       underlying hardware feels most comfortable in dealing with.

   2.  Use of tolerance types is only allowed for variables of a local scope;
       it is prohibited for public/external variables.  Writing the binary
       contents of a tolerance variable to a disk file is punishable by death.

   3.  Negative values are not allowed for type int. (thus it is 15 bits wide
       instead of 16 bits wide.)  Use type sint if you are planning to assign
       negative values to a variable.

   4.  Alternatively, type int can be prohibited by #defining it as "$@$#$"
       (in the end of this header file) and the use of uint or sint can be
       promoted instead.

   5.  Operations that result in overflow/underflow beyond the minimum
       number of guaranteed bits yield undefined results and should be
       avoided for all tolerance types.
*/
```

FIG. 17B

```
1
2   /* --- Portable types ------------------------------------------------------ */
3
4   typedef                 char  int8 ;
5   typedef signed          char sint8 ;
6   typedef unsigned        char uint8 ;
7   typedef           short int   int16;
8   typedef signed    short int  sint16;
9   typedef unsigned  short int  uint16;
10  typedef           long  int   int32;
11  typedef signed    long  int  sint32;
12  typedef unsigned  long  int  uint32;
13
14  typedef unsigned        char  flg8 ;
15  typedef unsigned short int   flg16;
16  typedef unsigned long  int   flg32;
17
18  typedef uint32                _ctx;          // max. size between _datap & uint32
19  #define NO_CTX ((_ctx)0)
20
21
22  typedef uint32                _id32;        // 32-bit id
23  #define NO_ID32  ((_id32)0)
24
25  typedef uint16                _id16;        // 16-bit id
26  #define NO_ID16  ((_id16)0)
27
28  typedef _id32                 _id;          // default id (32-bit)
29  #define NO_ID    ((_id)0)
30
31
32  typedef int8                  bool8;         // the shortest stand-alone boolean
33
34  #define MININT8   (( int8)  0x00)
35  #define MAXINT8   (( int8)  0x7F)
36
37  #define MINSINT8  ((sint8)  0x80)
38  #define MAXSINT8  ((sint8)  0x7F)
39
40  #define MINUINT8  ((uint8)  0x00)
41  #define MAXUINT8  ((uint8)  0xFF)
42
43  #define MININT16  (( int16) 0x0000)
44  #define MAXINT16  (( int16) 0x7FFF)
45
46  #define MINSINT16 ((sint16) 0x8000)
47  #define MAXSINT16 ((sint16) 0x7FFF)
48
49  #define MINUINT16 ((uint16) 0x0000U)
50  #define MAXUINT16 ((uint16) 0xFFFFU)
51
52  #define MININT32  (( int32) 0x00000000L)
53  #define MAXINT32  (( int32) 0x7FFFFFFFL)
54
55  #define MINSINT32 ((sint32) 0x80000000L)
56  #define MAXSINT32 ((sint32) 0x7FFFFFFFL)
57
58  #define MINUINT32 ((uint32) 0x00000000UL)
59  #define MAXUINT32 ((uint32) 0xFFFFFFFFUL)
```

FIG. 17C

```
1
2   /*
3     Notes:
4
5     1.  Use int16 and int32 when you know the exact bit width that you want
6         for your variable, but you do not care about the interpretation of
7         the sign of the variable.  This way, the compiler will choose an
8         actual signed or unsigned type, depending on what the underlying
9         hardware feels most comfortable in dealing with.
10
11    2.  As an example, variables that are ideal candidates to be declared as
12        int16 or int32 are sets of bitflags: although the bit capacity of such
13        a variable is crucial, the variable will never be interpreted as a
14        signed or unsigned value.
15
16    3.  All integer operations defined by the C language are valid on all of
17        the portable types, and overflow/underflow operations beyond
18        the specified bit widths have predictable behaviour.
19  */
20
21  #endif // defined _TYPES_DEFINED
22
```

FIG. 17D

```
1   /* ------------------------------------------------------------------------ */
2   /*                        ETC: Global system definitions                    */
3   /*                                                                          */
4   /*                       STATUS.H - global system statae                    */
5   /* ------------------------------------------------------------------------ */
6
7   #ifndef _STATUS_DEFINED
8   #define _STATUS_DEFINED
9
10  /* --- define status type ------------------------------------------------- */
11
12  #define _stat uint
13
14  /* --- define if_xxx macros ----------------------------------------------- */
15
16  #define if_ret(s,e)       if (((s) = (e)) != ST_OK) return ((s))
17  #define if_cleanup(s,e,c) if (((s) = (e)) != ST_OK) return (((c), (s)))
18
19  /* --- define general status values --------------------------------------- */
20
21  #define ST_OK              0 // successful operation
22
23  /* structures/tables manipulation */
24  #define ST_ALLOC          20 // can't alloc (for variable size alloc/pool)
25  #define ST_NO_ROOM        21 // can't alloc fixed sz entry in fixed sz pool
26  #define ST_OVERFLOW       22 // value, buffer, whatever may overflow
27  #define ST_UNDERFLOW      23 // value, buffer, counter, whatever may underflow
28  #define ST_EMPTY          24 // QUEUE/smth is empty - no elements
29  #define ST_FULL           25 // queue/smth is full
30  #define ST_EOF            26 // end of file
31
32  /* valchk */
33  #define ST_INVALID        30 // value not valid in operation context
34  #define ST_BAD_VALUE      31 // value not valid in module's context
35  #define ST_OUT_OF_RANGE   32 // value out of range
36  #define ST_NULL_PTR       33 // NULL ptr is not valid in operation context
37  #define ST_BAD_SYNTAX     34 // text/formatted data syntax error
38  #define ST_BAD_NAME       35 // name invalid in module's context
39
40  /* BAD errors */
41  #define ST_UNEXPECTED     40 // unexpected condition of parm or external object
42  #define ST_PANIC          41 // unexpected condition of self or internal object
43  #define ST_DEADLOCK       42 // deadlock detected
44
45  /* failures */
46  #define ST_REFUSE         45 // oper rejected voluntarily (inappropriate state)
47  #define ST_NO_ACTION      46 // oper requires no action (eg close closed file)
48  #define ST_FAILED         47 // operation failed / not successful
49
50  /* state check */
51  #define ST_NOT_INITED     50 // object is not initialized
52  #define ST_NOT_ACTIVE     51 // oper allowed only when object is active
53  #define ST_NOT_OPEN       52 // oper allowed only when object is open
54
55  /* i/o system */
56  #define ST_IOERR          60 // I/O error - file system / peripherals error
57  #define ST_BAD_CHKSUM     61 // wrong chksum
58
59  /* set-related */
```

FIG. 18A

```
#define ST_NOT_FOUND        62 // requested item not found
#define ST_DUPLICATE        63 // duplicated item is not allowed

/* critical section / privileges */
#define ST_BUSY             70 // in critical section (dynamic, short period)
#define ST_ACCESS_DENIED    71 // temporary lack of access (dynamic, long period)
#define ST_PRIVILEGE        72 // privilege violation (static)
#define ST_SCOPE_VIOLATION  73 // (no comment)
#define ST_BAD_ACCESS       74 // access type not appropriate / not possible

/* operation status */
#define ST_PENDING          80 // operation is pending
#define ST_TIMEOUT          81 // timeout has expired
#define ST_CANCELLED        82 // operation cancelled on user's request
#define ST_ABORTED          83 // op. aborted on system/module's request
#define ST_RESET            84 // object has been orderly reset
#define ST_CLEANUP          85 // object is about to be destroyed, now in cleanup
#define ST_OVERRIDE         86 // operation/entity was overriden

/* API */
#define ST_CANT_BIND        90 // can't bind / resolve name, handle, etc.
#define ST_FPI_ERROR        91 // invalid function request, etc.
#define ST_WRONG_VERSION    92 // incompatible version of data/code detected
#define ST_NOT_IMPLEMENTED  93 // feature not implemented
#define ST_NOT_SUPPORTED    94 // partial implementation; feature not supported

/* environment-specific */
#define ST_BAD_UID         100 // OBJ:invalid uid
#define ST_BAD_MSG         101 // OBJ:invalid msg or object doesn't handle it
#define ST_OK_CUT          102 // OBJ:don't call more virt. objs; return ST_OK
#define ST_NOT_INHERITED   103 // OBJ:method was abstract; needs to be inherited
#define ST_BAD_PID         104 // OBJ:invalid pid
#define ST_WIN_DFLT        210 // WIN:WindowProc: call DefWinProc, use its result
#define ST_BLOCK_OVERRIDE  300 // KER:attempt to block a thread that is already
blocked

/* --- String table ------------------------------------------------------ */

#ifdef _ERR_STAT

struct STAT_TXT_S
   {
   _stat  s;
   char  *p;
   } ;

#define DEF_STATUS(s)          { (s), #s }

static struct STAT_TXT_S _err_sts_tbl[] =
   {
      DEF_STATUS ( ST_OK                    ),
      DEF_STATUS ( ST_ALLOC                 ),
      DEF_STATUS ( ST_NO_ROOM               ),
      DEF_STATUS ( ST_OVERFLOW              ),
      DEF_STATUS ( ST_UNDERFLOW             ),
      DEF_STATUS ( ST_EMPTY                 ),
      DEF_STATUS ( ST_FULL                  ),
      DEF_STATUS ( ST_EOF                   ),
      DEF_STATUS ( ST_INVALID               ),
```

FIG. 18B

```
        DEF_STATUS ( ST_BAD_VALUE              ),
        DEF_STATUS ( ST_OUT_OF_RANGE           ),
        DEF_STATUS ( ST_NULL_PTR               ),
        DEF_STATUS ( ST_BAD_SYNTAX             ),
        DEF_STATUS ( ST_BAD_NAME               ),
        DEF_STATUS ( ST_UNEXPECTED             ),
        DEF_STATUS ( ST_PANIC                  ),
        DEF_STATUS ( ST_DEADLOCK               ),
        DEF_STATUS ( ST_REFUSE                 ),
        DEF_STATUS ( ST_NO_ACTION              ),
        DEF_STATUS ( ST_FAILED                 ),
        DEF_STATUS ( ST_NOT_INITED             ),
        DEF_STATUS ( ST_NOT_ACTIVE             ),
        DEF_STATUS ( ST_NOT_OPEN               ),
        DEF_STATUS ( ST_IOERR                  ),
        DEF_STATUS ( ST_BAD_CHKSUM             ),
        DEF_STATUS ( ST_NOT_FOUND              ),
        DEF_STATUS ( ST_DUPLICATE              ),
        DEF_STATUS ( ST_BUSY                   ),
        DEF_STATUS ( ST_ACCESS_DENIED          ),
        DEF_STATUS ( ST_PRIVILEGE              ),
        DEF_STATUS ( ST_SCOPE_VIOLATION        ),
        DEF_STATUS ( ST_BAD_ACCESS             ),
        DEF_STATUS ( ST_PENDING                ),
        DEF_STATUS ( ST_TIMEOUT                ),
        DEF_STATUS ( ST_CANCELLED              ),
        DEF_STATUS ( ST_ABORTED                ),
        DEF_STATUS ( ST_RESET                  ),
        DEF_STATUS ( ST_CLEANUP                ),
        DEF_STATUS ( ST_OVERRIDE               ),
        DEF_STATUS ( ST_CANT_BIND              ),
        DEF_STATUS ( ST_FPI_ERROR              ),
        DEF_STATUS ( ST_WRONG_VERSION          ),
        DEF_STATUS ( ST_NOT_IMPLEMENTED        ),
        DEF_STATUS ( ST_NOT_SUPPORTED          ),
        DEF_STATUS ( ST_BAD_UID                ),
        DEF_STATUS ( ST_BAD_MSG                ),
        DEF_STATUS ( ST_OK_CUT                 ),
        DEF_STATUS ( ST_NOT_INHERITED          ),
        DEF_STATUS ( ST_BAD_PID                ),
        DEF_STATUS ( ST_WIN_DFLT               ),
        DEF_STATUS ( ST_BLOCK_OVERRIDE         )
    };

#endif  // _ERR_STAT


/* --- status codes 1000 and up are system- or module- specific ------------ */

#endif  // _STATUS_DEFINED
```

FIG. 18C

```
1  /* ------------------------------------------------------------------------ */
2  /*                        ETC: Global system definitions                    */
3  /*                                                                          */
4  /*                         IFACE.H - Interface support                      */
5  /*                                                                          */
6  /*      Copyright (c) 1994-95 Object Dynamics Corp. All Rights Reserved.    */
7  /* ------------------------------------------------------------------------ */
8
9  #ifndef _IFACE_DEFINED
10 #define _IFACE_DEFINED
11
12 /* --- type definitions ---------------------------------------------------- */
13
14 typedef uint _iid;
15
16 #define NO_IID 0
17
18 #define IID_COMPAT(i1,i2)  \
19    ((i1) == (i2)       ||  \
20     (i1) == IID_I_POLY ||  \
21     (i2) == IID_I_POLY )
22
23 /* --- support macro definitions ------------------------------------------- */
24
25 class _COMPONENT; // forward
26
27 class _INTERFACE
28    {
29    public:
30       const _iid         iid;
31       const _COMPONENT  *const objp;
32
33    public:
34       _INTERFACE (_iid id, _COMPONENT *op) : iid (id), objp (op) { }
35
36    protected:
37       virtual ~_INTERFACE (void) { } // (hack) - adds vtbl - simplifies ptr casts
38 #define __IFACE_OFFSET_OP0 1          // 1st operation starts at index 1 in vtbl
39    } ;                              // Keep in sync!
40
41 #define INTERFACE(i)              \
42    class i : public _INTERFACE \
43       {                          \
44       public:                    \
45          i (_COMPONENT *op) : _INTERFACE (IID_##i, op) { } \
46       protected:                 \
47          i (void)  : _INTERFACE (0, NULL) { } \
48          i (i& ir) : _INTERFACE (0, NULL) { } \
49          virtual ~i (void) { }
50
51 #define END_INTERFACE          \
52       } ;
53
54 #define CONSTANTS    public:
55 #define BUSES        public:
56 #define OPERATIONS   public:
57 #define METHODS      public:
58
59 #define DEFINE_BUS  \
```

FIG. 19A

```
1     struct BUS        \
2        {
3
4  #define BUS(x)       \
5      struct x         \
6         {
7
8  #define END_BUS      \
9        } ;
10
11 #define INTERFACE_BUS(x) \
12    typedef x BUS
13
14 #define DEFAULTS(x) \
15    BUS (void) { x; }
16
17 #define BUS_DEFAULTS(b,x) \
18    b (void) { x; }
19
20 #define FACTORY_DEFAULTS(x) \
21    FACTORY_BUS (void) { x; }
22
23 #define OP(o)  \
24    virtual _stat o (BUS *bp) = 0
25
26 #endif // _IFACE_DEFINED
```

FIG. 19B

```
1   /* ------------------------------------------------------------------------ */
2   /*                          IFC: Project Interfaces                         */
3   /*                                                                          */
4   /*                      I_TERMINAL.H - Terminal control                     */
5   /*                                                                          */
6   /*      Copyright (c) 1994-95 Object Dynamics Corp. All Rights Reserved.    */
7   /* ------------------------------------------------------------------------ */
8
9   #ifndef _I_TERMINAL_DEFINED
10  #define _I_TERMINAL_DEFINED
11
12
13  BUS ( B_TERMINAL )
14     // terminal identification
15     char          *namep  ;   // terminal name
16     uint           name_sz;   // size of name buffer
17
18     // terminal information
19     uint           type   ;   // terminal type (physical mechanism + direction)
20     uint32         card   ;   // terminal cardinality
21     flg16          attr   ;   // terminal attributes
22
23     // connection data
24     _iid           iid    ;   // terminal interface id (contract id)
25     _INTERFACE    *ip     ;   // v-table interface
26
27     // connection identification
28     _id32          conn_id;   // connection id (as defined by the terminal)
29     _COMPONENT    *comp   ;   // pointer to component ($$EV_EVB, see notes)
30     _id            link_id;   // link id ($$EV_EVB, see notes)
31
32  END_BUS
33
34  #define IID_I_TERMINAL    0x03
35
36  INTERFACE (I_TERMINAL)
37
38     /* --------------------------------------------------------------------- */
39     CONSTANTS
40        enum TYPE
41           {
42           T_INPUT  = 0,
43           T_OUTPUT = 1,
44           } ;
45
46        enum ATTR
47           {
48           A_STDSIG   = (1UL << 14), // terminal for standard signal
49           A_FLOATING = (1UL << 15), // terminal is ok to be unconnected
50           } ;
51
52     /* --------------------------------------------------------------------- */
53     BUSES
54        INTERFACE_BUS ( B_TERMINAL );
55
56     /* --------------------------------------------------------------------- */
57     OPERATIONS
58          OP ( acquire        );
59          OP ( release        );
```

FIG 20A

```
1           OP ( connect        );
2           OP ( disconnect     );
3           OP ( get_info       );
4           OP ( get_context    );
5           OP ( set_context    );
6           OP ( qry_open       );
7           OP ( qry_get_first  );
8           OP ( qry_get_last   );
9           OP ( qry_get_next   );
10          OP ( qry_get_prev   );
11          OP ( qry_get_curr   );
12          OP ( qry_close      );
13
14  END_INTERFACE
15
16  /* declaration macro */
17
18  #define DECLARE_I_TERMINAL(c,l) \
19     IDEF (c, l, I_TERMINAL)      \
20        OPDECL ( acquire       ); \
21        OPDECL ( release       ); \
22        OPDECL ( connect       ); \
23        OPDECL ( disconnect    ); \
24        OPDECL ( get_info      ); \
25        OPDECL ( get_context   ); \
26        OPDECL ( set_context   ); \
27        OPDECL ( qry_open      ); \
28        OPDECL ( qry_get_first ); \
29        OPDECL ( qry_get_last  ); \
30        OPDECL ( qry_get_next  ); \
31        OPDECL ( qry_get_prev  ); \
32        OPDECL ( qry_get_curr  ); \
33        OPDECL ( qry_close     ); \
34     END_IDEF                    \
35     /* allow for semicolon */    \
36     typedef int c##_l_##_DECLARE_I_TERMINAL_DUMMY
37
38  // Descriptions:
39
40  // on    acquire
41  // in :  namep    - terminal name
42  //       comp     - component on the other side
43  //       link_id - link id
44  // out: (ip)      - if type is T_INPUT  - pointer to the terminal interface
45  //       type     - terminal type (I_TERMINAL::T_xxx)
46  //       iid      - expected interface id
47  //       conn_id - connection id assigned for this connection
48  // act:  acquire connection context
49  // s  :  ST_NOT_FOUND    - terminal not found
50  //       ST_NO_ROOM      - terminal cardinality exhausted
51
52  // on    release
53  // in :  namep    - terminal name
54  //       ip       - if type is T_OUTPUT - interface to which the output was
55  connected
56  //                  if type is T_INPUT  - interface to release
57  //                  ($$ may not be needed and in most cases can not be provided)
58  //       comp     - component on the other side
59  //       link_id - link id
```

FIG 20B

```
//       conn_id - connection id assigned by this terminal on acquire
// out:  void
// act:  release terminal
// s  :  ST_NO_ACTION   - terminal was not connected
//                        (to/via specified interface or component)
//       ST_NOT_FOUND   - terminal not found

// on    connect
// in :  namep   - terminal name
//       type    - terminal type (I_TERMINAL::T_xxx)
//       iid     - expected interface id
//      (ip)     - if type is T_OUTPUT - interface to connect to
//       comp    - component to connect to
//       link_id - link id
//       conn_id - connection id assigned by this terminal on acquire
// out:  void
// act:  connect terminal
// s  :  ST_REFUSE      - interface mismatch (e.g., unacceptable IID)
//       ST_NOT_FOUND   - terminal not found

// on    disconnect
// in :  namep    - terminal name
//       ip       - if type is T_OUTPUT - interface to which the output is
connected
//                  if type is T_INPUT  - interface to disconnect
//                  ($$ may not be needed and in most cases can not be provided)
//       comp     - component to disconnect from
//       link_id  - link id
//       conn_id - connection id assigned by this terminal on acquire
// out:  void
// act:  disconnect terminal
// s  :  ST_NO_ACTION   - terminal was not connected
//                        (to/via specified interface or component)

// on    get_info
// in :  namep - terminal name
// out:  type  - terminal type (I_TERMINAL::T_xxx)
//       iid   - interface id of terminal
//       card  - terminal cardinality (static, not current)
//       attr  - terminal attributes
// act:  return information about specified terminal
// s  :  ST_NOT_FOUND   - terminal not found
// nb :  the information returned by this operation is not affected by
//       the current state of the terminal

// on    get_context
// not implemented

// on    set_context
// not implemented

// on    qry_open
// in :  namep    - query string
// out:  ctx      - query context for subsequent qry_xxx operations
// act:  open query on terminal namespace
// s  :  ST_NO_ROOM    - too many open queries
//       ST_BAD_SYNTAX - bad query syntax

// on    qry_get_first
```

FIG 20C

```
1   // in :  namep     - buffer for name or NULL
2   //      (name_sz) - size of buffer, [bytes]
3   //       ctx      - query context from previous qry_xxx operation
4   // out:(*namep)   - terminal name
5   //       ctx      - query context for subsequent qry_xxx operation
6   // act:  get first matching terminal name
7   // s  :  ST_NOT_FOUND   - no matching terminals
8
9   // on    qry_get_last
10  // in :  namep     - buffer for name or NULL
11  //      (name_sz) - size of buffer, [bytes]
12  //       ctx      - query context from previous qry_xxx operation
13  // out:(*namep)   - terminal name
14  //       ctx      - query context for subsequent qry_xxx operation
15  // act:  get last matching terminal name
16  // s  :  ST_NOT_FOUND   - no matching terminals
17
18  // on    qry_get_next
19  // in :  namep     - buffer for name or NULL
20  //      (name_sz) - size of buffer, [bytes]
21  //       ctx      - query context from previous qry_xxx operation
22  // out:(*namep)   - terminal name
23  //       ctx      - query context for subsequent qry_xxx operation
24  // act:  get next matching terminal name
25  // s  :  ST_NOT_FOUND   - no more matching terminals
26
27  // on    qry_get_prev
28  // in :  namep     - buffer for name or NULL
29  //      (name_sz) - size of buffer, [bytes]
30  //       ctx      - query context from previous qry_xxx operation
31  // out:(*namep)   - terminal name
32  //       ctx      - query context for subsequent qry_xxx operation
33  // act:  get previous matching terminal name
34  // s  :  ST_NOT_FOUND   - no more matching terminals
35
36  // on    qry_get_curr
37  // in :  namep     - buffer for name or NULL
38  //      (name_sz) - size of buffer, [bytes]
39  //       ctx      - query context from previous qry_xxx operation
40  // out:(*namep)   - terminal name
41  // act:  get current terminal name in query
42  // nb :  ctx is unchanged
43
44  // on    qry_close
45  // in :  ctx      - query context from qry_open or another qry_xxx operation
46  // out:  void
47  // act:  close query on terminal name space
48
49  // $$EV_EVB -- fields used by the event bus terminal interface
50  // Note: comp in the bus is provided for use by connectivity components only.
51  //       It should not be used by regular components & assemblies.
52  //       comp contains pointer identifying the component at the other side
53  //       of the link.
54  //       Both link_id and comp may be used for identification of the particular
55  //       connection on a terminal. They should not be interpreted in any way
56  //       nor the comp should be dereferenced. Sometimes the ip field on
57  //       disconnect and release may be used for connection identification, too.
58
59  // Note: the assembly manager (dflt assembly implementation) does not store
```

FIG 20D

```
1   //        the connection id -- so it may not be provided on disconnect/release
2
3   // Note: $$ on disconnect ip may not be provided in many cases:
4   //        a) when output array is connected to input array
5   //        b) when assy output is connected to input array
6   //        In both examples above the input ptr cannot be obtained from the output
7   //        and therefore cannot be passed on input's disconnect.
8   //        (Generally, none of the existing parts uses ip for identification)
9
10  #endif // _I_TERMINAL_DEFINED
```

FIG 20E

```
1   /* ------------------------------------------------------------------------ */
2   /*                          IFC: Project Interfaces                         */
3   /*                                                                          */
4   /*                            I_PROP.H - Property                           */
5   /*                                                                          */
6   /*      Copyright (c) 1994-95 Object Dynamics Corp. All Rights Reserved.    */
7   /* ------------------------------------------------------------------------ */
8
9   #ifndef _I_PROP_DEFINED
10  #define _I_PROP_DEFINED
11
12  /* --- .type values */
13
14  #define PT_USER         0               // user defined
15  #define PT_INT          1               // signed int
16  #define PT_UINT         2               // unsigned int
17  #define PT_CHAR         3               // uchar
18  #define PT_UPCHAR       4               // upcaseduchar
19  #define PT_STRING       5               // asciiz, no multiple spaces
20  #define PT_NAME         6               // upcased STRING
21  #define PT_DATA         7               // data string (0x1b,'-0', etc.)
22  #define PT_NONE         8               // 1st invalid value of .type
23
24
25  BUS ( B_PROP )
26
27     char    *namep  ;
28     size_t  name_sz;
29     uint32  type    ;
30     void    *p      ;
31     size_t  sz      ;
32     size_t  len     ;
33     _ctx    ctx     ;
34
35  END_BUS
36
37  #define IID_I_PROP    0x02
38
39  INTERFACE (I_PROP)
40
41     /* ------------------------------------------------------------------- */
42     CONSTANTS
43
44     /* ------------------------------------------------------------------- */
45     BUSES
46        INTERFACE_BUS ( B_PROP );
47
48     /* ------------------------------------------------------------------- */
49     OPERATIONS
50          OP ( get            );
51          OP ( set            );
52          OP ( chk            );
53          OP ( get_info       );
54          OP ( qry_open       );
55          OP ( qry_get_first  );
56          OP ( qry_get_last   );
57          OP ( qry_get_next   );
58          OP ( qry_get_prev   );
59          OP ( qry_get_curr   );
```

FIG. 21A

```
        OP ( qry_close      );

END_INTERFACE

/* declaration macro */

#define DECLARE_I_PROP(c,l)      \
   IDEF (c, l, I_PROP)           \
      OPDECL ( get           ); \
      OPDECL ( set           ); \
      OPDECL ( chk           ); \
      OPDECL ( get_info      ); \
      OPDECL ( qry_open      ); \
      OPDECL ( qry_get_first ); \
      OPDECL ( qry_get_last  ); \
      OPDECL ( qry_get_next  ); \
      OPDECL ( qry_get_prev  ); \
      OPDECL ( qry_get_curr  ); \
      OPDECL ( qry_close     ); \
   END_IDEF                     \
   /* allow for semicolon */    \
   typedef int c##_l_##_DECLARE_I_PROP_DUMMY

// --- Descriptions

// note: on get, set and chk - if value type is specified and it doesn't match
the
//       internal type of the property, the component may (implementation-
defined)
//       convert the value to the appropriate type;

// on    get
// in :  namep    - property name
//       p        - buffer for value or NULL
//      (sz)      - size of buffer (if p not NULL), [bytes]
//       type     - requested type or IP_NONE for any
// out:(*p)       - value
//       len      - value length (for both fixed & var size types), [bytes]
//       type     - property type
// act:  get property value
// s  :  ST_BAD_NAME    - property name not found
//       ST_OVERFLOW    - buffer too small
//       ST_REFUSE      - incorrect type
//       ST_BAD_ACCESS  - property is write-only

// on    set
// in :  namep    - property name
//       p        - buffer for value or NULL to clear
//      (len)     - size of value (for var size values), [bytes]
//       type     - type or IP_NONE if not known
// out:  void
// act:  set property value
// s  :  ST_BAD_NAME    - property name not found
//       ST_BAD_VALUE   - bad property value
//       ST_OVERFLOW    - value too long
//       ST_REFUSE      - incorrect type
//       ST_BAD_ACCESS  - property is read-only

// on    chk
```

FIG. 21B

```
1   // in :  namep     - property name
2   //       p         - buffer for value or NULL to clear
3   //      (len)      - size of value (for var size values), [bytes]
4   //       type      - value type or IP_NONE if not known
5   // out:  void
6   // act:  check if property may be set to value (valchk)
7   // s  :  ST_BAD_NAME     - property name not found
8   //       ST_BAD_VALUE    - bad property value
9   //       ST_OVERFLOW     - value too long
10  //       ST_REFUSE       - incorrect type
11  //       ST_BAD_ACCESS   - property is read-only
12
13
14  // on    get_info
15  // in :  namep     - property name
16  // out:  type      - property type
17  // act:  get information about specified property
18  // s  :  ST_BAD_NAME     - property name not found
19  // nb :  the information returned by this operation is not affected by
20  //       the current value of the property
21
22  // on    qry_open
23  // in :  namep     - query string
24  // out:  ctx       - query context for subsequent qry_xxx operations
25  // act:  open query on property space
26  // s  :  ST_NO_ROOM     - too many open queries
27  //       ST_BAD_SYNTAX - bad query syntax
28
29  // on    qry_get_first
30  // in :  namep     - buffer for name or NULL
31  //      (name_sz)  - size of buffer, [bytes]
32  //       ctx       - query context from previous qry_xxx operation
33  // out:(*namep)    - property name
34  //       ctx       - query context for subsequent qry_xxx operation
35  // act:  get first matching property name
36  // s  :  ST_NOT_FOUND    - no matching properties
37
38  // on    qry_get_last
39  // in :  namep     - buffer for name or NULL
40  //      (name_sz)  - size of buffer, [bytes]
41  //       ctx       - query context from previous qry_xxx operation
42  // out:(*namep)    - property name
43  //       ctx       - query context for subsequent qry_xxx operation
44  // act:  get last matching property name
45  // s  :  ST_NOT_FOUND    - no matching properties
46
47  // on    qry_get_next
48  // in :  namep     - buffer for name or NULL
49  //      (name_sz)  - size of buffer, [bytes]
50  //       ctx       - query context from previous qry_xxx operation
51  // out:(*namep)    - property name
52  //       ctx       - query context for subsequent qry_xxx operation
53  // act:  get next matching property name
54  // s  :  ST_NOT_FOUND    - no more matching properties
55
56  // on    qry_get_prev
57  // in :  namep     - buffer for name or NULL
58  //      (name_sz)  - size of buffer, [bytes]
59  //       ctx       - query context from previous qry_xxx operation
```

FIG. 21C

```
1   // out:(*namep)    - property name
2   //       ctx       - query context for subsequent qry_xxx operation
3   // act:  get previous matching property name
4   // s  :  ST_NOT_FOUND   - no more matching properties
5
6   // on    qry_get_curr
7   // in :  namep     - buffer for name or NULL
8   //      (name_sz)  - size of buffer, [bytes]
9   //       ctx       - query context from previous qry_xxx operation
10  // out:(*namep)    - property name
11  // act:  get current property name in query
12  // nb :  ctx is unchanged
13
14  // on    qry_close
15  // in :  ctx       - query context from qry_open or another qry_xxx operation
16  // out:  void
17  // act:  close query on property space
18
19  #endif // _I_PROP_DEFINED
20
```


FIG. 21D

```
/* ------------------------------------------------------------------------ */
/*                          IFC: Project Interfaces                         */
/*                                                                          */
/*              I_FACTORY.H - Component creation / destruction              */
/*                                                                          */
/*      Copyright (c) 1994-95 Object Dynamics Corp. All Rights Reserved.    */
/* ------------------------------------------------------------------------ */

#ifndef  _I_FACTORY_DEFINED
#define  _I_FACTORY_DEFINED


#define IID_I_FACTORY    0x01

INTERFACE (I_FACTORY)

   /* --------------------------------------------------------------------- */
   CONSTANTS
      enum
         {
         A_PRIMARY   = (1UL << 14),       // create primary object (factory)
         A_NO_ALLOC  = (1UL << 13),       // don't call new to alloc memory
         A_GENERIC   = (1UL << 12),       // generic factory bus (no ext. data)
         } ;

   /* --------------------------------------------------------------------- */
   BUSES
      DEFINE_BUS
         flg32                          attr;       // factory attributes
         _COMPONENT                    *objp;       // ptr to component or memory
         _INTERFACE                    *termp;      // terminal interface
         _INTERFACE                    *propp;      // property interface
         _ctx                           ctx  ;      // creation context or NO_CTX
      END_BUS

   /* --------------------------------------------------------------------- */
   OPERATIONS
      OP ( create_dflts  );
      OP ( create        );
      OP ( destroy       );
      OP ( get_part_ifc  );

END_INTERFACE

/* declaration macro */

#define DECLARE_I_FACTORY(c,l)          \
   IDEF (c, l, I_FACTORY)               \
      OPDECL (create        );          \
      OPDECL (create_dflts  );          \
      OPDECL (destroy       );          \
      OPDECL (get_part_ifc  );          \
   END_IDEF                             \
   /* allow for semicolon */            \
   typedef int c##_l_##_DECLARE_I_FACTORY_DUMMY

/* --- Descriptions ------------------------------------------------------- */

// on    create_dflts   (struct BUS or derived)
```

FIG. 22A

```
1   // in :  void
2   // out:  attr  - default attributes
3   //       objp  - NULL
4   // act:  fill FACTORY_BUS with defaults
5
6   // on    create          (struct BUS or derived)
7   // in :  attr  - object attributes (A_xxx)
8   //      (objp) - ptr to allocated memory for object instance (if A_NO_ALLOC)
9   //       ctx   - additional context or NO_CTX
10  // out:  objp  - ptr to new object
11  //       termp - pointer to the terminal interface of the part
12  //       propp - pointer to the property interface of the part
13  // act:  create new object
14  // nb :  if A_GENERIC is set, then the bus passed is strictly I_FACTORY::BUS and
15  //       not a derived one.
16  // nb :  ctx==NO_CTX is always accepted; some components may allow additional
17  //       parameters in/through this field
18
19  // on    destroy         (struct BUS or derived)
20  // in :  void
21  // out:  void
22  // act:  destroy object
23
24  // on    get_part_ifc  (struct BUS or derived)
25  // in :  void
26  // out:  termp - pointer to the terminal interface of the part
27  //       propp - pointer to the property interface of the part
28  // act:  return part interfaces
29
30  #endif // _I_FACTORY_DEFINED
```

FIG. 22B

```
1   /* ------------------------------------------------------------------------ */
2   /*                              CLP: Collections Parts                      */
3   /*                                                                          */
4   /*                          CP_BBA.CPP - Bag on Byte Array                  */
5   /*                                                                          */
6   /* Copyright (c) 1993-1994 Object Dynamics Corp. All Rights Reserved.       */
7   /* ------------------------------------------------------------------------ */
8
9   #include <rtx_pre.h>
10  #include <stddef.h>
11  #include <stdlib.h>
12  #include <memory.h>
13  #include <rtx_post.h>
14
15  #include <rtx.h>
16  #include <asm.h>
17  #include <thp.h>
18
19  #include <iface.h>
20  #include <part.h>
21
22  #include <i_factor.h>
23  #include <i_prop.h>
24  #include <i_drain.h>
25  #include <i_bag.h>
26  #include <i_agm.h>
27  #include <i_bytear.h>
28
29  /* --- Definitions ------------------------------------------------------- */
30
31  struct CP_BBA_ENTRY_HDR
32     {
33     enum ATTR
34        {
35        A_VALID     = VALID,
36        A_FREE      = FREE,
37        A_KILLED    = KILLED,
38        };
39
40     flg32          attr;          // attributes [A_VALID, A_FREE, A_KILLED]
41     uint32         card;          // cardinality
42     };
43
44  /** --- Component Definition ---------------------------------------------- */
45
46
47  #define CP_BBA_NAME  "CP_BBA"
48
49  PART ( CP_BBA )
50
51     /* ------------------------------------------------------------------- */
52     BUSES
53        DEFAULT_FACTORY_BUS ( CP_BBA );
54
55     /* ------------------------------------------------------------------- */
56     INPUTS
57        IDECL ( CP_BBA , factory    , I_FACTORY     );
58        IDECL ( CP_BBA , term       , I_TERMINAL    );
59        IDECL ( CP_BBA , prop       , I_PROP        );
```

FIG. 23A

```
1          IDECL ( CP_BBA , bag          , I_BAG          );
2
3          IDECL_RST (CP_BBA);
4
5       OUTPUTS
6          ODECL ( CP_BBS , arr          , I_BYTEARR      );
7
8
9    END_PART  // CP_BBA
10
11   #ifdef BIT32
12      #define hmemcpy memcpy
13      #define HUGE
14   #else
15      #define HUGE __huge
16   #endif
17
18   /* --- Definitions ------------------------------------------------------ */
19
20   #define AGM_CHUNK     1024
21
22   BEGIN_SELF ( CP_BBA )
23      // data
24      uint32          n_entries;     // number of entries
25      void           *buffp;         // buffer for one entry
26      uint32          buff_sz;       // buffer size (dep. on data_sz)
27
28      // properties
29      uint32          data_sz;        // data size
30      uint32          key_sz;         // key size
31      uint32          key_off;        // key offset
32      char            name [64];
33   END_SELF
34
35   BEGIN_CLASS_DATA ( CP_BBA )
36      uint16      instance_counter;          // ...
37   END_CLASS_DATA
38
39   /* --- Interface declarations ------------------------------------------- */
40
41   DECLARE_I_FACTORY  ( CP_BBA, factory );
42   DECLARE_I_TERMINAL ( CP_BBA, term    );
43   DECLARE_I_PROP     ( CP_BBA, prop    );
44   DECLARE_I_BAG      ( CP_BBA, bag     );
45
46   DECLARE_RST        (CP_BBA);
47
48   /* --- I/O tables ------------------------------------------------------- */
49
50   INPUT_TABLE
51      INPUT     (CP_BBA, prop    , I_PROP    )
52      INPUT     (CP_BBA, bag     , I_BAG     )
53
54      INPUT_RST (CP_BBA)
55   END_INPUT_TABLE
56
57   OUTPUT_TABLE
58      OUTPUT    (CP_BBA, arr     , I_BYTEARR )
59
```

FIG. 23B

```
END_OUTPUT_TABLE

/* --- Constants & tables ------------------------------------------------ */

PROPERTIES
   /* type    name       base  field       min    max    */

   PROP_UINT   ("DATA_SIZE"          , SELF, data_sz    , 1     , 1024     ),
   PROP_UINT   ("KEY_SIZE"           , SELF, key_sz     , 1     , 1024     ),
   PROP_UINT   ("KEY_OFFSET"         , SELF, key_off    , 0     , 1023     ),

   PROP_STRING ("name"               , SELF, name       , 0     , 64       ),
END_PROPERTIES

/* --- Init / Cleanup ---------------------------------------------------- */

INIT (CP_BBA)
   {
   /** init class data */
   cp->instance_counter = 0;

   REGISTER (CP_BBA);

   return (ST_OK);
   } /* INIT (CP_BBA) */

CLEANUP (CP_BBA)
   {
   DEREGISTER (CP_BBA);

   return (ST_OK);
   } /* CLEANUP (CP_BBA) */

/* --- Constructors  ----------------------------------------------------- */

DEFAULT_PART_CONSTRUCTORS (CP_BBA);


/* --- Operations: factory ----------------------------------------------- */

DEFAULT_PART_FACTORY  (CP_BBA, factory);
DEFAULT_PART_TERMINAL (CP_BBA, term   );
DEFAULT_PART_PROP     (CP_BBA, prop   );

/* --- Operations -------------------------------------------------------- */

METHOD (CP_BBA, create_dflts , FACTORY_BUS)
   {

   // ...

   return (ST_OK);
   } END_METHOD /* CP_BBA, create_dflts    */

METHOD (CP_BBA, create , FACTORY_BUS)
   {
   _stat   s;

   /** valchk */
```

FIG. 23C

```
1       if (bp == NULL) return (ST_NULL_PTR);
2
3       /** open interfaces */
4       OPEN_INTERFACE (CP_BBA, factory    );
5       if (bp->attr & I_FACTORY::A_PRIMARY) return (ST_OK);   // valid primary obj
6       OPEN_INTERFACE (CP_BBA, term       );
7       OPEN_INTERFACE (CP_BBA, prop       );
8       OPEN_INTERFACE (CP_BBA, bag        );
9       OPEN_RST       (CP_BBA);
10
11      /** init self */
12      sp->n_entries = 0;
13      sp->buffp     = NULL;
14      sp->buff_sz   = 0;
15      sp->data_sz   = 2;
16      sp->key_sz    = 2;
17      sp->key_off   = 0;
18
19      /** internal reset */
20      (void)call (sp->thisp->iRST, raise, NULL);
21
22      /** special processing for default name */
23      os_con_sprintf (sp->name, "%.50s%d", CP_BBA_NAME, ++(cp->instance_counter));
24
25      return (ST_OK);
26      } END_METHOD /* CP_BBA, create */
27
28  METHOD (CP_BBA, destroy, FACTORY_BUS)
29      {
30      /** close interfaces */
31      CLOSE_INTERFACE (CP_BBA, factory    );
32      CLOSE_INTERFACE (CP_BBA, term       );
33      CLOSE_INTERFACE (CP_BBA, prop       );
34      CLOSE_INTERFACE (CP_BBA, bag        );
35      CLOSE_RST       (CP_BBA);
36
37      /** free entry buffer */
38      if (sp->buffp != NULL) (void)os_smp_free (sp->buffp);
39
40      /** init self */
41      sp->n_entries = 0;
42      sp->buffp     = NULL;
43      sp->buff_sz   = 0;
44      sp->data_sz   = 2;
45
46      return (ST_OK);
47      } END_METHOD /* CP_BBA, destroy */
48
49  /* --- Operations: bag ------------------------------------------------------ */
50
51  OPERATION ( CP_BBA, bag, add      )
52      {
53      I_BYTEARR::BUS     ab;
54      uint32             i;
55      CP_BBA_ENTRY_HDR  *hdrp;
56      byte              *datap;
57      _stat              s;
58
59      /** valchk */
```

FIG. 23D

```
1       if (bp == NULL) return (ST_NULL_PTR);
2       if (bp->datap == NULL) return (ST_INVALID);
3
4       /** internal init */
5       hdrp  = (CP_BBA_ENTRY_HDR  *)sp->buffp;
6       datap = (byte *)sp->buffp + sizeof (*hdrp);
7
8       /** --- lookup */
9       for (i = 0; i < sp->n_entries; i++)
10         {
11         /** read entry */
12         ab.p    = sp->buffp;
13         ab.sz   = sp->buff_sz;
14         ab.pos  = i * sp->buff_sz;
15         ab.len  = sp->buff_sz;
16         ab.attr = I_BYTEARR::A_EXACT;
17         s = out (arr, read, &ab);
18         if (s == ST_EOF) return (ST_UNEXPECTED);
19         if_ret (s, s);
20
21         /** skip if empty */
22         if ((hdrp->attr == CP_BBA_ENTRY_HDR::A_FREE)  ||
23             (hdrp->attr == CP_BBA_ENTRY_HDR::A_KILLED)) continue;
24
25         /** skip if different */
26         if (memcmp ((void *)(datap + sp->key_off),
27                     (void *)((byte *)bp->datap + sp->key_off),
28                     sp->key_sz) != 0) continue;
29
30         /** sanity chk */
31         if (hdrp->card == 0) return (ST_UNEXPECTED);
32
33         /** increment cardinality */
34         if (hdrp->card < MAXUINT32 - 1) (hdrp->card)++;
35         else return (ST_OVERFLOW);
36
37         /** write back header only */
38         ab.p    = hdrp;
39         ab.len  = sizeof (*hdrp);
40         ab.pos  = i * sp->buff_sz;
41         ab.attr = 0;
42         s = out (arr, write, &ab);
43         if (s == ST_OVERFLOW) return (ST_UNEXPECTED);
44         if_ret (s, s);
45
46         /** pass cardinality */
47         bp->card = hdrp->card;
48
49         /** return */
50         return (ST_OK);
51         } // for
52
53      /** --- add  */
54      for (i = 0; i < sp->n_entries; i++)
55         {
56         /** read entry */
57         ab.p    = sp->buffp;
58         ab.sz   = sp->buff_sz;
59         ab.pos  = i * sp->buff_sz;
```

FIG. 23E

```
        ab.len  = sp->buff_sz;
        ab.attr = I_BYTEARR::A_EXACT;
        s = out (arr, read, &ab);
        if (s == ST_EOF) return (ST_UNEXPECTED);
        if_ret (s, s);

        /** write if empty */
        if ((hdrp->attr == CP_BBA_ENTRY_HDR::A_FREE) != 0)
           {
           /** init header */
           hdrp->card = 1;
           hdrp->attr = CP_BBA_ENTRY_HDR::A_VALID;

           /** init data */
           memcpy ((void *)datap, bp->datap, sp->data_sz);

           /** write header and data */
           ab.p    = sp->buffp;
           ab.len  = sp->buff_sz;
           ab.pos  = i * sp->buff_sz;
           ab.attr = 0;
           s = out (arr, write, &ab);
           if (s == ST_OVERFLOW) return (ST_UNEXPECTED);
           if_ret (s, s);

           /** pass cardinality */
           bp->card = hdrp->card;

           /** return */
           return (ST_OK);
           }
        } // for

     /** chk for overflow */
     if (sp->n_entries >= MAXUINT32 - 1) return (ST_NO_ROOM);

     /** init header */
     hdrp->card  = 1;
     hdrp->attr = CP_BBA_ENTRY_HDR::A_VALID;

     /** init data */
     memcpy ((void *)datap, bp->datap, sp->data_sz);

     /** append */
     ab.p    = sp->buffp;
     ab.len  = sp->buff_sz;
     ab.pos  = sp->n_entries * sp->buff_sz;
     ab.attr = I_BYTEARR::A_GROW;
     s = out (arr, write, &ab);
     if (s == ST_OVERFLOW) return (ST_UNEXPECTED);
     if_ret (s, s);

     /** increment number of entries */
     (sp->n_entries)++;

     /** pass cardinality */
     bp->card =  hdrp->card;

     return (ST_OK);
```

FIG. 23F

```
1     } END_OPERATION /* bag, add     */
2
3   // TO DO: other operation of I_BAG to be implemented here…
4
5   /* --- Signals ------------------------------------------------------------ */
6
7   SIGNAL_RST ( CP_BBA )
8     {
9     I_AGM::BUS          ab;
10    _stat               s;
11
12    /** free entry buffer */
13    if (sp->buffp != NULL) (void)os_smp_free (sp->buffp);
14
15    /** init self */
16    sp->n_entries = 0;
17    sp->buffp     = NULL;
18    sp->buff_sz   = sizeof (CP_BBA_ENTRY_HDR) + max (sp->data_sz, 1);
19
20    /** alloc entry buffer */
21    if_ret (s, os_smp_alloc (sp->buff_sz, &(sp->buffp)));
22
23    return (ST_OK);
24    } END_SIGNAL /* RST */
25
26
```

FIG. 23G

```
1   /* ------------------------------------------------------------------------ */
2   /*                          EVP: Event Flow Parts                           */
3   /*                                                                          */
4   /*                      EV_PCN.CPP - POLY T-Connector                       */
5   /*                                                                          */
6   /*      Copyright (c) 1994-95 Object Dynamics Corp. All Rights Reserved.    */
7   /* ------------------------------------------------------------------------ */
8   /* Version 1.10                                                             */
9   /* ------------------------------------------------------------------------ */
10
11  #include <rtx_pre.h>
12  #include <stddef.h>
13  #include <stdlib.h>
14  #include <rtx_post.h>
15
16  #include <rtx.h>
17  #include <asm.h>
18
19  #include <iface.h>
20  #include <part.h>
21  #include <assembly.h>
22
23  #include <i_factor.h>
24  #include <i_prop.h>
25  #include <i_term.h>
26  #include <i_drain.h>
27  #include <i_poly.h>
28
29  #define EV_PCN_NAME  "EV_PCN"
30
31  ASSEMBLY (EV_PCN)
32
33     /* ------------------------------------------------------------------- */
34     BUSES
35        DEFAULT_FACTORY_BUS (EV_PCN);
36
37     /* ------------------------------------------------------------------- */
38     INPUTS
39        IDECL ( EV_PCN , factory    , I_FACTORY  );
40        IDECL ( EV_PCN , term       , I_TERMINAL );
41        IDECL ( EV_PCN , termi      , I_TERMINAL );
42        IDECL ( EV_PCN , prop       , I_PROP     );
43        IDECL ( EV_PCN , propi      , I_PROP     );
44        IDECL ( EV_PCN , in         , I_POLY     );
45
46     OUTPUTS
47        ODECL ( EV_PCN , out        , I_POLY     );
48        ODECL ( EV_PCN , aux        , I_DRAIN    );
49
50
51  END_ASSEMBLY
52
53
54  /* --- Definitions ------------------------------------------------------ */
55
56  BEGIN_SELF (EV_PCN)
57
58     DEFAULT_ASM_SELF;
59
```

FIG. 24A

```
1     /* group properties */
2     uint32      ev_id_min;
3     uint32      ev_id_max;
4
5   END_SELF
6
7   BEGIN_CLASS_DATA (EV_PCN)
8     // none
9   END_CLASS_DATA
10
11  DECLARE_ASM (EV_PCN);
12
13  /* --- I/O tables ------------------------------------------------------- */
14
15  INPUT_TABLE
16    INPUT     (EV_PCN, prop    , I_PROP    )
17    INPUT     (EV_PCN, in      , I_POLY    )
18  END_INPUT_TABLE
19
20  OUTPUT_TABLE
21    OUTPUT_I  (EV_PCN, out     , I_POLY    )
22    OUTPUT_I  (EV_PCN, aux     , I_DRAIN   )
23  END_OUTPUT_TABLE
24
25  /* --- Constants & tables ----------------------------------------------- */
26
27  ASM_PARTS
28    NEW_PART ( p2d, EV_P2D )
29    NEW_PART ( tcn, EV_TCN )
30    NEW_PART ( d2p, EV_D2P )
31  END_ASM_PARTS
32
33  ASM_PARAMETRIZATION
34    // none
35  END_ASM_PARAMETRIZATION
36
37  ASM_LINKS
38    //          from output          to input
39
40    LINK      ( *   , in            , p2d, in   )
41    LINK      ( p2d, out           , tcn, in   )
42    LINK      ( tcn, out           , d2p, in   )
43    LINK      ( d2p, out           , *  , out  )
44    LINK      ( tcn, aux           , *  , aux  )
45
46  END_ASM_LINKS
47
48  ASM_PROPERTY_GROUP ( ev_id_min_group  )
49    ASM_PROP_MEMBER ( tcn  , OUT.EV_ID_MIN )
50    ASM_PROP_MEMBER ( tcn  , AUX.EV_ID_MIN )
51  END_ASM_PROPERTY_GROUP
52
53  ASM_PROPERTY_GROUP ( ev_id_max_group  )
54    ASM_PROP_MEMBER ( tcn  , OUT.EV_ID_MAX )
55    ASM_PROP_MEMBER ( tcn  , AUX.EV_ID_MAX )
56  END_ASM_PROPERTY_GROUP
57
58  ASM_PROPERTIES
59    ASM_PROP_ALIAS  ( OUT.INVERT    , tcn , OUT.INVERT     )
```

FIG. 24B

```
1      ASM_PROP_ALIAS  ( AUX.INVERT     , tcn , AUX.INVERT     )
2
3      ASM_PROP_GROUP  ( EV_ID_MIN      , ev_id_min_group      )
4      ASM_PROP_GROUP  ( EV_ID_MAX      , ev_id_max_group      )
5   END_ASM_PROPERTIES
6
7   ASM_GROUP_PROPERTY_DESC
8
9      /* type     name         base  field       min          max    */
10
11     PROP_UINT ("EV_ID_MIN", SELF, ev_id_min , MINUINT32 , MAXUINT32 ),
12     PROP_UINT ("EV_ID_MAX", SELF, ev_id_max , MINUINT32 , MAXUINT32 ),
13
14  END_ASM_GROUP_PROPERTY_DESC
15
16  DEFAULT_ASM (EV_PCN);
17
```

FIG. 24C

```
1  /* ------------------------------------------------------------------------ */
2  /*                          IFC: Project Interfaces                         */
3  /*                                                                          */
4  /*                           I_POLY.H - Polymorphic                         */
5  /*                                                                          */
6  /*      Copyright (c) 1994-95 Object Dynamics Corp. All Rights Reserved.    */
7  /* ------------------------------------------------------------------------ */
8
9  #ifndef _I_POLY_DEFINED
10 #define _I_POLY_DEFINED
11
12 #define IID_I_POLY    0x04
13
14 INTERFACE (I_POLY)
15
16    /* ------------------------------------------------------------------------ */
17    CONSTANTS
18
19    /* ------------------------------------------------------------------------ */
20    BUSES
21         DEFINE_BUS
22          // none
23         END_BUS
24
25    /* ------------------------------------------------------------------------ */
26    OPERATIONS
27         OP ( op0          );
28         OP ( op1          );
29         OP ( op2          );
30         OP ( op3          );
31         OP ( op4          );
32         OP ( op5          );
33         OP ( op6          );
34         OP ( op7          );
35         OP ( op8          );
36         OP ( op9          );
37         OP ( op10         );
38         OP ( op11         );
39         OP ( op12         );
40         OP ( op13         );
41         OP ( op14         );
42         OP ( op15         );
43         OP ( op16         );
44         OP ( op17         );
45         OP ( op18         );
46         OP ( op19         );
47         OP ( op20         );
48         OP ( op21         );
49         OP ( op22         );
50         OP ( op23         );
51         OP ( op24         );
52         OP ( op25         );
53         OP ( op26         );
54         OP ( op27         );
55         OP ( op28         );
56         OP ( op29         );
57         OP ( op30         );
58         OP ( op31         );
59         OP ( op32         );
```

FIG. 25A

```
1           OP ( op33        );
2           OP ( op34        );
3           OP ( op35        );
4           OP ( op36        );
5           OP ( op37        );
6           OP ( op38        );
7           OP ( op39        );
8           OP ( op40        );
9           OP ( op41        );
10          OP ( op42        );
11          OP ( op43        );
12          OP ( op44        );
13          OP ( op45        );
14          OP ( op46        );
15          OP ( op47        );
16          OP ( op48        );
17          OP ( op49        );
18          OP ( op50        );
19          OP ( op51        );
20          OP ( op52        );
21          OP ( op53        );
22          OP ( op54        );
23          OP ( op55        );
24          OP ( op56        );
25          OP ( op57        );
26          OP ( op58        );
27          OP ( op59        );
28          OP ( op60        );
29          OP ( op61        );
30          OP ( op62        );
31          OP ( op63        );
32
33  END_INTERFACE
34
35  /* declaration macro */
36
37  #define DECLARE_I_POLY(c,1)              \
38     IDEF (c, 1, I_POLY)                   \
39        OPDECL ( op0          );           \
40        OPDECL ( op1          );           \
41        OPDECL ( op2          );           \
42        OPDECL ( op3          );           \
43        OPDECL ( op4          );           \
44        OPDECL ( op5          );           \
45        OPDECL ( op6          );           \
46        OPDECL ( op7          );           \
47        OPDECL ( op8          );           \
48        OPDECL ( op9          );           \
49        OPDECL ( op10         );           \
50        OPDECL ( op11         );           \
51        OPDECL ( op12         );           \
52        OPDECL ( op13         );           \
53        OPDECL ( op14         );           \
54        OPDECL ( op15         );           \
55        OPDECL ( op16         );           \
56        OPDECL ( op17         );           \
57        OPDECL ( op18         );           \
58        OPDECL ( op19         );           \
59        OPDECL ( op20         );           \
```

FIG. 25B

```
        OPDECL ( op21          );          \
        OPDECL ( op22          );          \
        OPDECL ( op23          );          \
        OPDECL ( op24          );          \
        OPDECL ( op25          );          \
        OPDECL ( op26          );          \
        OPDECL ( op27          );          \
        OPDECL ( op28          );          \
        OPDECL ( op29          );          \
        OPDECL ( op30          );          \
        OPDECL ( op31          );          \
        OPDECL ( op32          );          \
        OPDECL ( op33          );          \
        OPDECL ( op34          );          \
        OPDECL ( op35          );          \
        OPDECL ( op36          );          \
        OPDECL ( op37          );          \
        OPDECL ( op38          );          \
        OPDECL ( op39          );          \
        OPDECL ( op40          );          \
        OPDECL ( op41          );          \
        OPDECL ( op42          );          \
        OPDECL ( op43          );          \
        OPDECL ( op44          );          \
        OPDECL ( op45          );          \
        OPDECL ( op46          );          \
        OPDECL ( op47          );          \
        OPDECL ( op48          );          \
        OPDECL ( op49          );          \
        OPDECL ( op50          );          \
        OPDECL ( op51          );          \
        OPDECL ( op52          );          \
        OPDECL ( op53          );          \
        OPDECL ( op54          );          \
        OPDECL ( op55          );          \
        OPDECL ( op56          );          \
        OPDECL ( op57          );          \
        OPDECL ( op58          );          \
        OPDECL ( op59          );          \
        OPDECL ( op60          );          \
        OPDECL ( op61          );          \
        OPDECL ( op62          );          \
        OPDECL ( op63          );          \
    END_IDEF                               \
    /* allow for semicolon */              \
    typedef int c##_1_##_DECLARE_I_POLY_DUMMY

#endif // _I_POLY_DEFINED
```

FIG. 25C

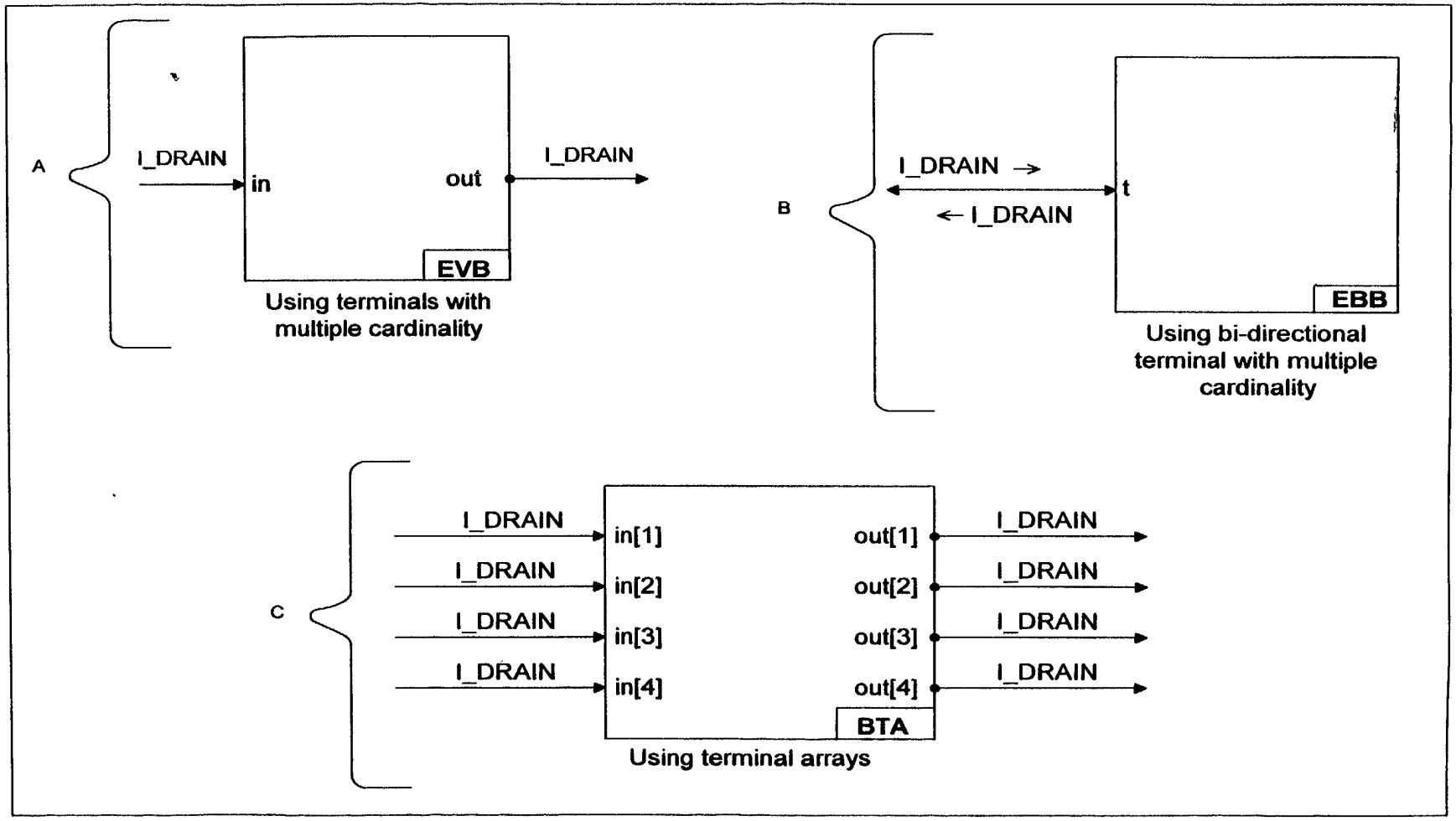

A
I_DRAIN
in
out
I_DRAIN
EVB
Using terminals with multiple cardinality
B
I_DRAIN →
← I_DRAIN
t
EBB
Using bi-directional terminal with multiple cardinality
C
I_DRAIN
I_DRAIN
I_DRAIN
I_DRAIN
in[1]
in[2]
in[3]
in[4]
out[1]
out[2]
out[3]
out[4]
I_DRAIN
I_DRAIN
I_DRAIN
I_DRAIN
BTA
Using terminal arrays

FIG. 26

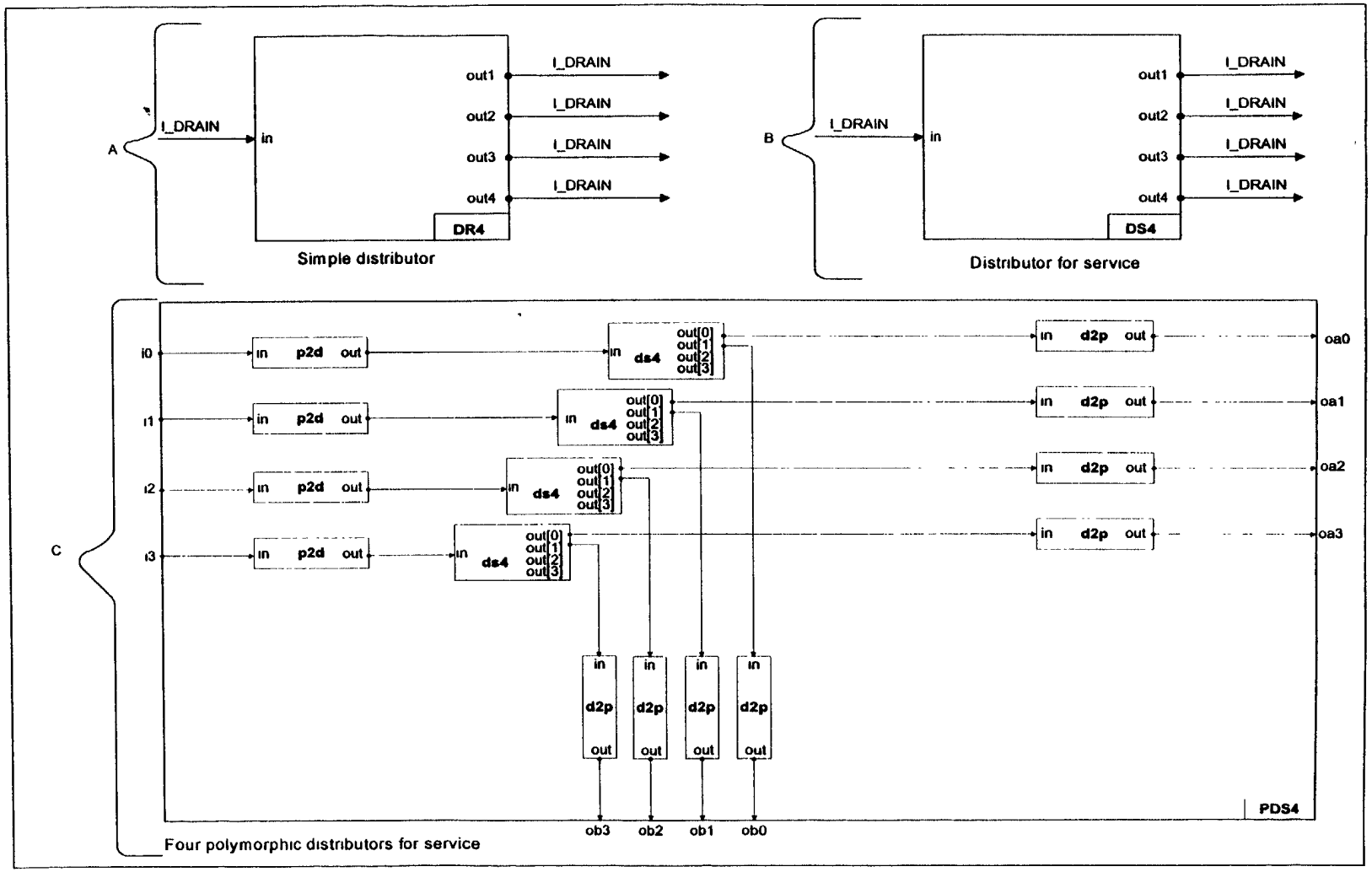

A
I_DRAIN
in
out1
out2
out3
out4
I_DRAIN
I_DRAIN
I_DRAIN
I_DRAIN
DR4
Simple distributor
B
I_DRAIN
in
out1
out2
out3
out4
I_DRAIN
I_DRAIN
I_DRAIN
I_DRAIN
DS4
Distributor for service
C
i0
i1
i2
i3
in p2d out
in ds4 out[0] out[1] out[2] out[3]
in d2p out
oa0
oa1
oa2
oa3
ob3
ob2
ob1
ob0
PDS4
Four polymorphic distributors for service

FIG. 27

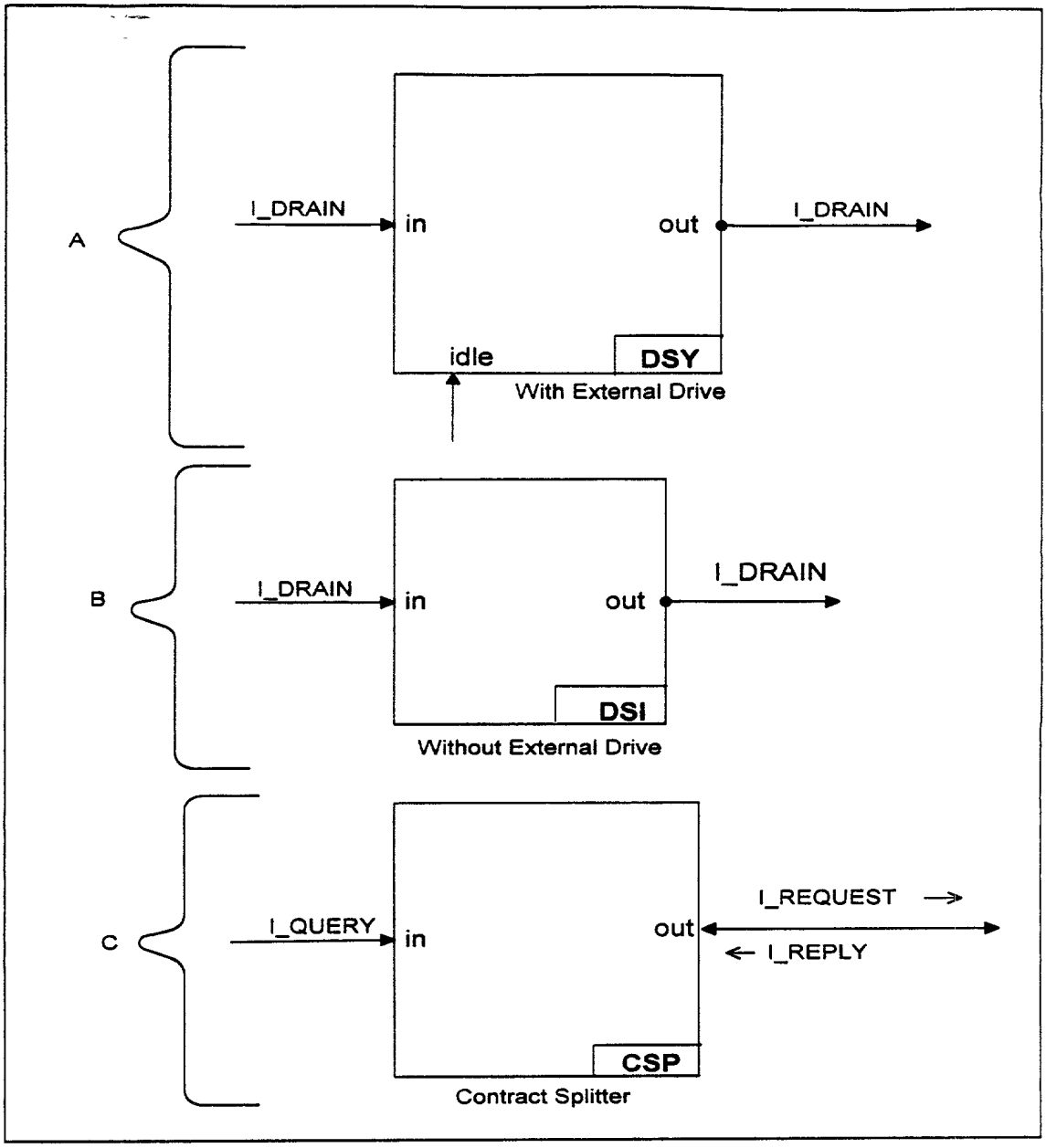

A
I_DRAIN
in
out
I_DRAIN
idle
DSY
With External Drive
B
I_DRAIN
in
out
I_DRAIN
DSI
Without External Drive
C
I_QUERY
in
out
I_REQUEST
I_REPLY
CSP
Contract Splitter

FIG. 28

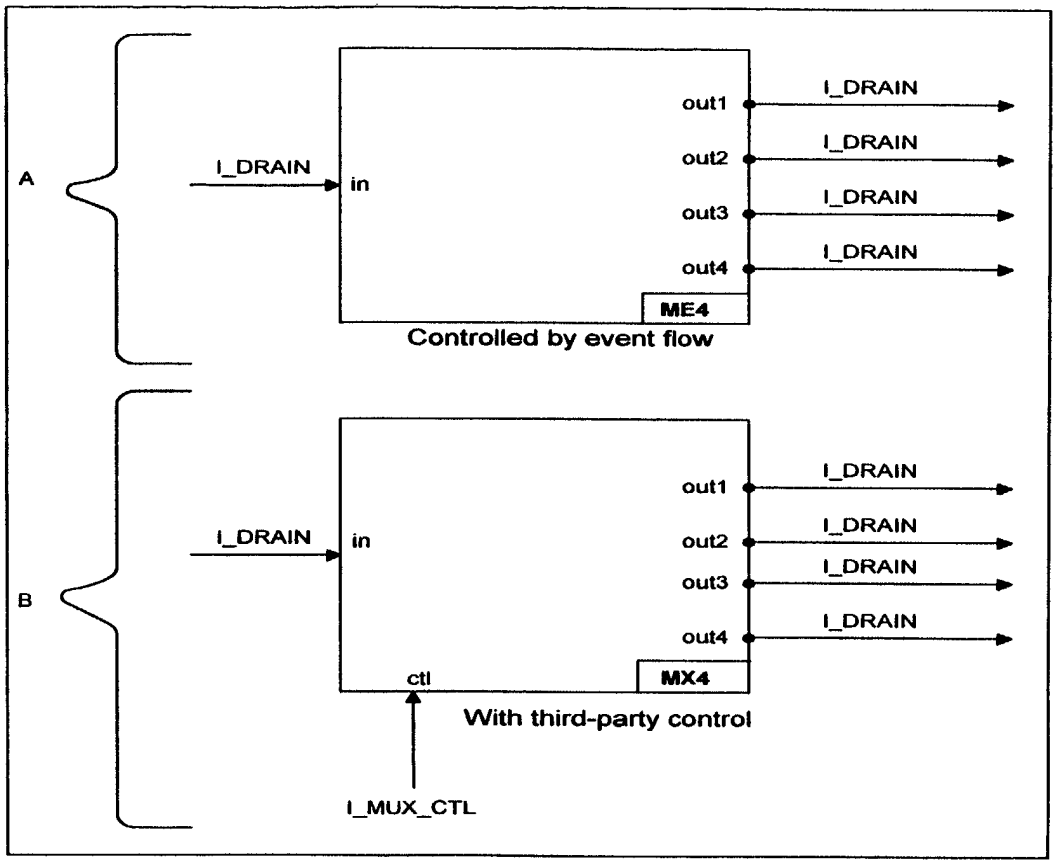

A
I_DRAIN
in
out1
out2
out3
out4
I_DRAIN
I_DRAIN
I_DRAIN
I_DRAIN
ME4
Controlled by event flow
B
I_DRAIN
in
out1
out2
out3
out4
I_DRAIN
I_DRAIN
I_DRAIN
I_DRAIN
ctl
MX4
With third-party control
I_MUX_CTL

FIG. 29

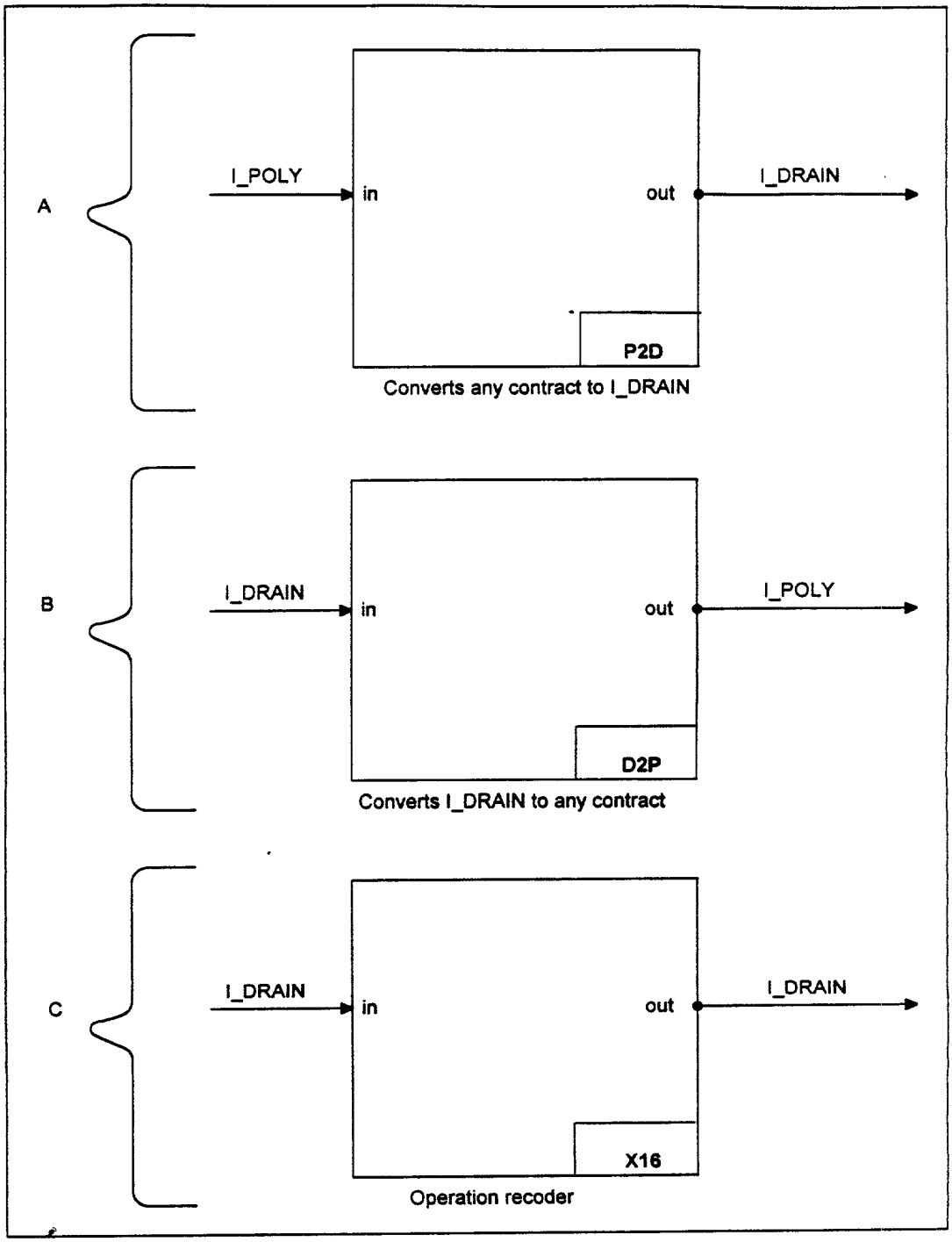

A
I_POLY
in
out
I_DRAIN
P2D
Converts any contract to I_DRAIN
B
I_DRAIN
in
out
I_POLY
D2P
Converts I_DRAIN to any contract
C
I_DRAIN
in
out
I_DRAIN
X16
Operation recoder

FIG. 30

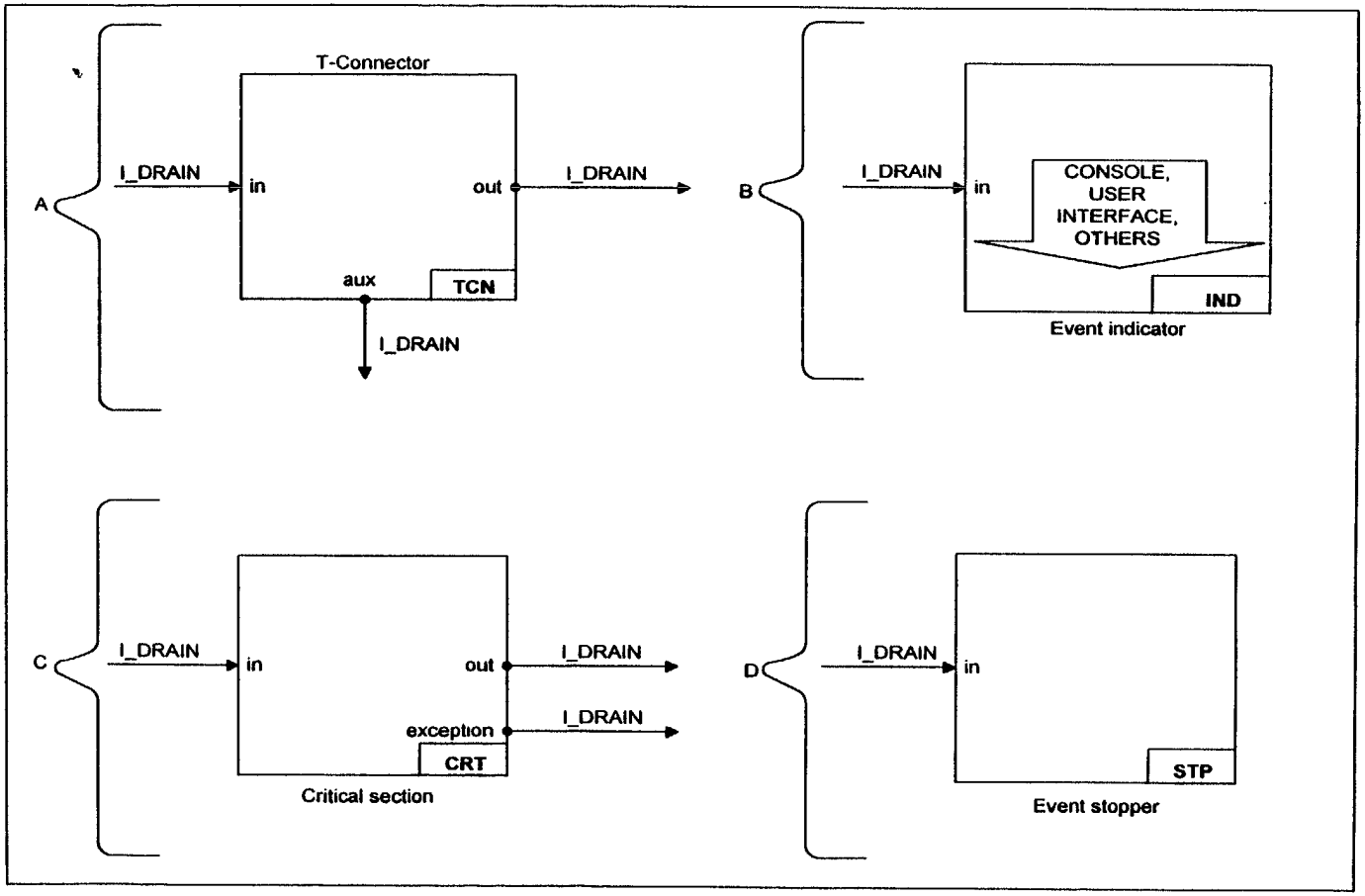

T-Connector
A
I_DRAIN
in
out
I_DRAIN
aux
TCN
I_DRAIN
B
I_DRAIN
in
CONSOLE,
USER
INTERFACE,
OTHERS
IND
Event indicator
C
I_DRAIN
in
out
I_DRAIN
exception
I_DRAIN
CRT
Critical section
D
I_DRAIN
in
STP
Event stopper

FIG. 31

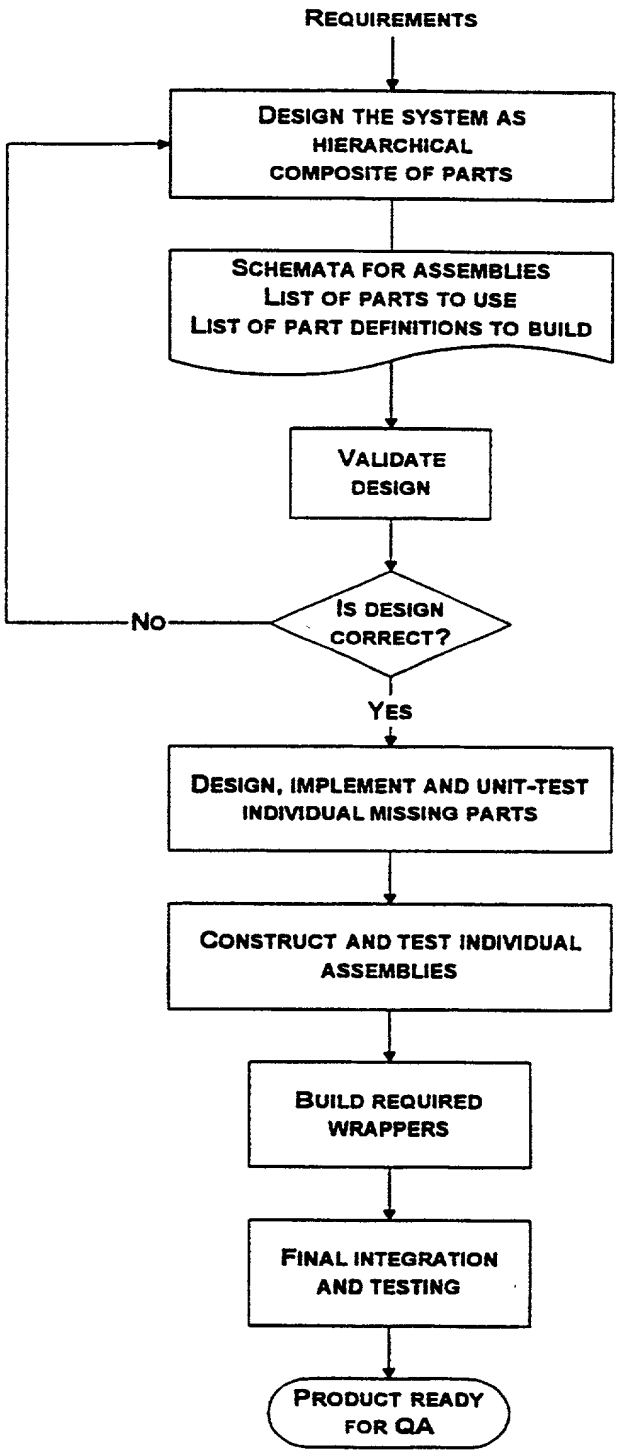

REQUIREMENTS
DESIGN THE SYSTEM AS HIERARCHICAL COMPOSITE OF PARTS
SCHEMATA FOR ASSEMBLIES
LIST OF PARTS TO USE
LIST OF PART DEFINITIONS TO BUILD
VALIDATE DESIGN
IS DESIGN CORRECT?
NO
YES
DESIGN, IMPLEMENT AND UNIT-TEST INDIVIDUAL MISSING PARTS
CONSTRUCT AND TEST INDIVIDUAL ASSEMBLIES
BUILD REQUIRED WRAPPERS
FINAL INTEGRATION AND TESTING
PRODUCT READY FOR QA

FIG. 32

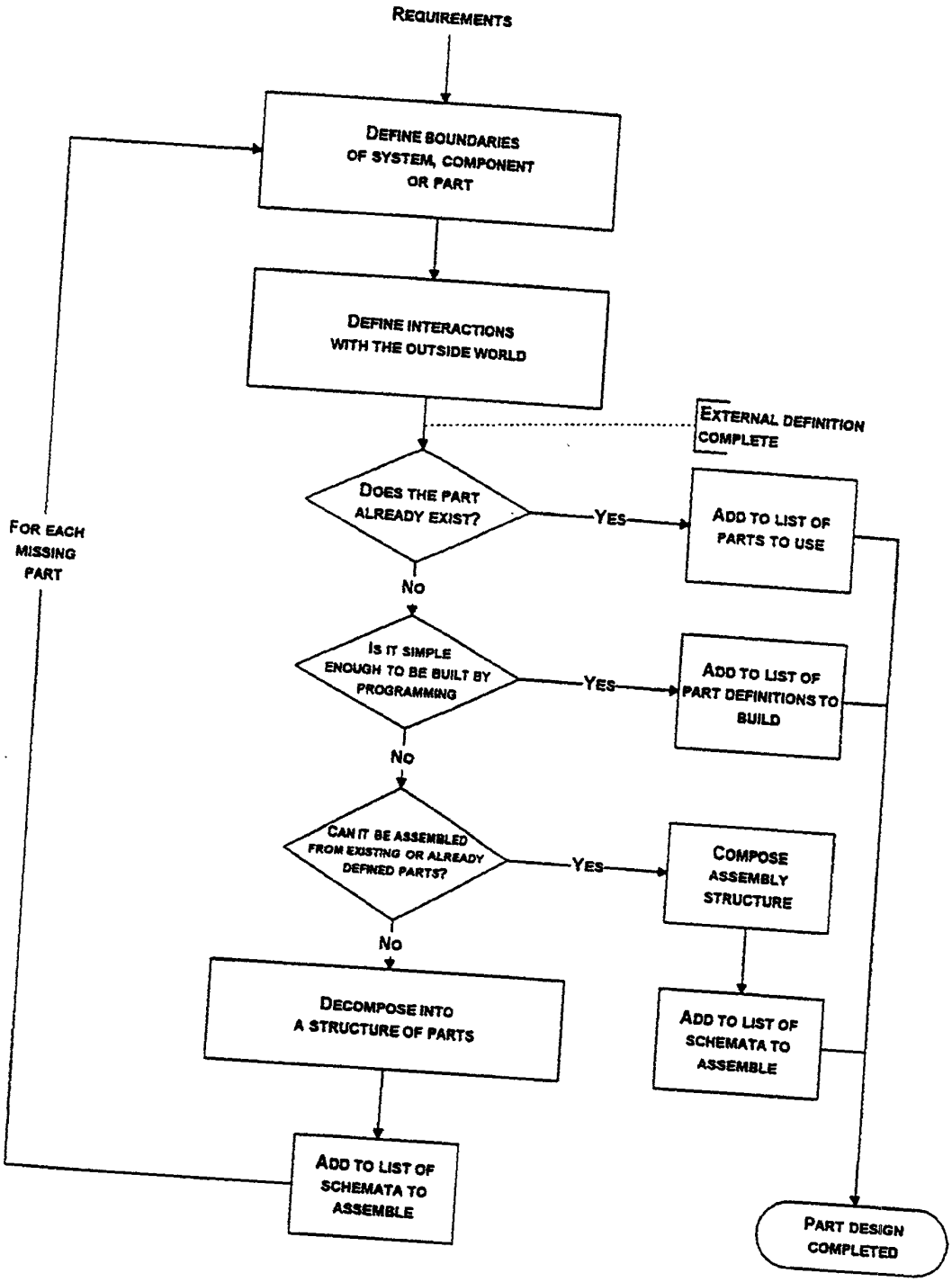

REQUIREMENTS
DEFINE BOUNDARIES OF SYSTEM, COMPONENT OR PART
DEFINE INTERACTIONS WITH THE OUTSIDE WORLD
EXTERNAL DEFINITION COMPLETE
DOES THE PART ALREADY EXIST?
YES
ADD TO LIST OF PARTS TO USE
NO
IS IT SIMPLE ENOUGH TO BE BUILT BY PROGRAMMING
YES
ADD TO LIST OF PART DEFINITIONS TO BUILD
NO
CAN IT BE ASSEMBLED FROM EXISTING OR ALREADY DEFINED PARTS?
YES
COMPOSE ASSEMBLY STRUCTURE
NO
DECOMPOSE INTO A STRUCTURE OF PARTS
ADD TO LIST OF SCHEMATA TO ASSEMBLE
FOR EACH MISSING PART
ADD TO LIST OF SCHEMATA TO ASSEMBLE
PART DESIGN COMPLETED

FIG. 33

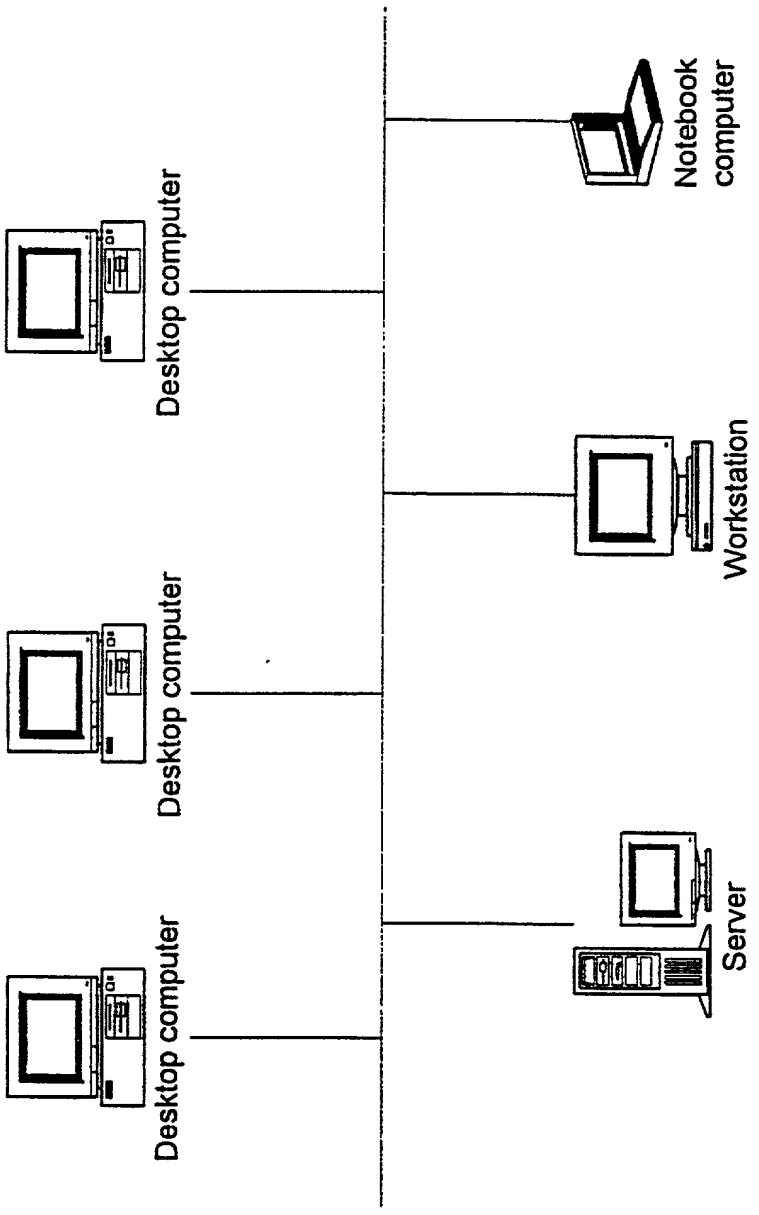

Desktop computer
Desktop computer
Desktop computer
Notebook computer
Workstation
Server

FIG. 34